ASG Software Solutions

ASG-Manager Products™ Relational Technology Support: DB2

Version 2.5.2

Publication Number: MPR0200-252-RELDB
Publication Date: August 2002

The information contained herein is the confidential and proprietary information of Allen Systems Group, Inc. Unauthorized use of this information and disclosure to third parties is expressly prohibited. This technical publication may not be reproduced in whole or in part, by any means, without the express written consent of Allen Systems Group, Inc.

© 1998-2002 Allen Systems Group, Inc. All rights reserved.
All names and products contained herein are the trademarks or registered trademarks of their respective holders.

ASG Worldwide Headquarters Naples Florida USA | asg.com | info@asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 239.435.2200 Fax: 239.263.3692 Toll Free: 800.932.5536 (USA only)

Contents

Preface vii

About this Publication vii

Publication Conventions viii

ASG Customer Support xi
Intelligent Support Portal (ISP) xi
Telephone Support xi

ASG Documentation/Product Enhancement Fax Form xiii

1 Introduction 1

Features: The Tools Provided 2
Corporate Repository 3
Diagramming Functions 4
Data Modeling and Design Functions 4
Data Definition Language (DDL) Generator 5
COBOL, PL/I, and Assembler Language Generator 5
Dynamically Submitting SQL Statements to DB2 or SQL/DS 6
Importing Information from DB2 6

Functions: How to Use the Tools Provided 7
Standards 7
DB2 Database Design 8
Implementation 11
Maintenance 13
Summary 16

Benefits 17
A Shared and Reusable Corporate Model 17
Automated Design 19
Conclusion 20

2 What Do You Want To Do? 21

ASG Support for Your DB2 Environment 21

Designing a DB2 Database 21
DB2 Database Design 21
Producing Output Describing the DB2 Design 22

DB2 Repository Definition 22
Documenting a DB2 Dictionary Schema 22
DB2 Object Definition 22

Export to DB2 23
Generating Output 23
Tailoring Generated Output 24

Dynamically Submitting SQL Statements 25

Importing from DB2 25

3 DB2 Database Design 27

Introduction to DB2 Database Design 28
Support for Referential Integrity 30
Introduction to Referential Structures and Cycles 31
Features to Support DB2 31

Designing a DB2 Database 32
Creating Entity and Userview Models 32
Generating a Relational Schema 33
Generating the DB2 Design 34
Reporting the DB2 Design 35
Populating the Dictionary with DB2 Members 36
Examples of the DB2 Database Design Process 37

DB2 Command Output 54
Output from the DB2 REPORT Command 54
Output from the DB2 PLOT CLUSTER Command 63
Output from the DB2 PLOT REFERENTIAL-STRUCTURES Command 71
Output from the DB2 LIST TABLES Command 81
Output from the DB2 LIST CYCLES Command 83

Generated DB2 Member Definitions 85
Generated DB2-TABLE Member 85
Generated DB2-INDEX Member 86
Generated DB2-VIEW Member 87
Generated SYSTEM Member 88

4 Repository Definition 91

Introduction to Documenting a DB2 DBMS 91

Documenting DB2 Objects 92

Documenting the Columns of Indexes, Tables, and Views 94

Documenting DB2 Security Information 96

Naming Conventions for DB2 Members 97
Generating External Names 97
Naming Guidelines 101

Interrogating Your DB2 Dictionary Schema 102

5 Export to DB2 105

Generating Output 106
Submitting Generated Output to Your Relational Environment 107

Tailoring Output 109
Introduction to Tailoring 110
Generating Object Names and External Names from Aliases 114
Tailoring DATE and TIME Character Field Lengths 115
Generating a Host Language Data Structure with an SQL DECLARE Statement 116
Generating an SQL DECLARE Statement with a Host Language Data Structure 117
Setting the Release of DB2 117
Setting the Release Flag 117
Generating Flat or Nested Data Structures 117
Generating Indicator Structures 118
Generating Indicator Suffixes on Structures 118
Setting Suffixes Applied to Indicator Array Names 119
Setting Suffixes Applied to Variable-Length Column Names 119
Automatically Generating SQL COMMENT ON/LABEL ON Statements 119
Generating One-, Two-, or Three-part Names for DB2 Objects 120
Setting a Tolerance Level for Output 121
Setting the SQL Escape Character 121
Setting Width of Output for SQL COMMENT ON Statements 122
Setting Width/Indent of the SQL DROP Impact Analysis Report 122
Allowing an Optional Space Character when Generating SQL DECIMAL Datatypes 122
Accessing a Specific DB2 Subsystem or Plan 123
Setting EXPORT Generated Object-name Length 123
Setting the Generated Column Data Type 124
Creating an INSERT Statement for Stored Procedures 124
Introduction to User Exits 124
Taking User Exits when Accessing a Repository Member 125
Taking User Exits For Specified DB2 Export Functions 125
Taking User Exits for an Individual Export Function 129
Setting Switches for Non-standard SQL Generation 130

6 Dynamic Import/Export 131

Security and Authorization 133

Output 133

Using Executive Routines with Dynamic SQL Functions 134
Variables Used for Dynamic Import/Export 139
Control Variables 139
Return Variables 140
COMMAND and EXECUTIVE Members Used in Dynamic SQL Functions 141
Creating and Populating a Table 141
Inserting Rows into a Table 142
Importing Information and Assigning it to Command Variables 144
Submitting Any SQL Statement that Can Be Prepared 146
Creating Your Own HELP Text 147

7 Importing From DB2 149

Introduction 149
Naming Guidelines 152
Documenting Columns 155

Tailoring Import 157
Tailorable Corporate Executive Routines 158
Global Variables Used in the Import Commands 160

8 Commands 175

Command Descriptions 176
DB2 ALTER 176
DB2 BIND and DB2 REBIND 195
DB2 COMMENT and DB2 LABEL 206
DB2 CREATE 212
DB2 DEBUG 219
DB2 DECLARE 223
DB2 DROP 228
DB2 GRANT and DB2 REVOKE 234
DB2 LIST CYCLES 239
DB2 LIST TABLES 240
DB2 PLOT CLUSTER 242
DB2 PLOT REFERENTIAL-STRUCTURES 245
DB2 POPULATE 250
DB2 PREVIEW 262
DB2 PRODUCE 273
DB2 RECALCULATE 282
DB2-RENAME 285
DB2 REPORT 288

DB2 SIZE 291
DB2 SYNONYM 295
EXTRACT DB2 299
ISQL 306
POPULATE 309
PREVIEW IMPORT 312
RADD 316
RECONCILE 317
RIGN 329
RREN 330
RREP 331
RUPD 332

Output Generation Options 334
Sending Generated Output to a USER-MEMBER 334
Sending Generated Output to a Sequential Dataset 335
Sending Generated Output to a Partitioned Dataset 336
Sending Generated Output to PRINT 336
Examples of Output Generation Options 336

Name Editing Options. 337
Dropping or Replacing a Name. 338
Inserting a Character String Within a Name 338
Examples of Name Editing Options 339

9 Repository Member Types 341

Member Type Descriptions 342
DB2-ALIAS 342
DB2-AUXTABLE 347
DB2-COLLECTION. 350
DB2-DATABASE. 351
DB2-DMS 355
DB2-FUNCTION 363
DB2-INDEX 370
DB2-LOCATION 390
DB2-PACKAGE. 392
DB2-PLAN 398
DB2-PRIVILEGE 405
DB2-PROCEDURE 417
DB2-STOGROUP. 422
DB2-TABLE. 425
DB2-TBSPACE 456
DB2-TEMPTABLE 469
DB2-TRIGGER. 472
DB2-USER 476
DB2-VIEW 481

Reusing Existing Member Definitions. 509

Appendix A
Name Reduction Process 511

Description 511

Example 512

Appendix B
Documenting Other Relational Databases 515

Appendix C
Defining and Generating
DB2 Member Types 519

Relationship Between DB2 Member Types 519

Data Types Generated from Form Descriptions 522

Explanation 524
NUMERIC-CHARACTER Form-descriptions 526

Index 527

Preface

This *ASG-Manager Products Relational Technology Support: DB2* is one of a series from the ASG-Manager family of program products for organizations seeking to automate and manage their application development and maintenance effort.

This publication provides ASG-Manager Products support for the DB2 environment. It is assumed that you have basic understanding of ASG-Manager Products (herein called Manager Products) and you are familiar with the DB2 environment and DB2 technology.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "Introduction," gives you an introductory overview of the support provided by Manager Products for DB2, and lists the benefits gained from using such support.
- Chapter 2, "What Do You Want To Do?," directs you to the relevant documentation in this manual.
- Chapter 3, "DB2 Database Design," describes how to automate the production of a first-cut DB2 database design and populate your repository with relevant member definitions.
- Chapter 4, "Repository Definition," relates how to document a DB2 environment in your repository.
- Chapter 5, "Export to DB2," describes how to generate output to implement and maintain your DB2 environment, and how to export output to DB2.

- Chapter 6, "Dynamic Import/Export," explains how to dynamically submit generated output to your DB2 environment, and receive the results, from within Manager Products.
- Chapter 7, "Importing From DB2," outlines how to import information about objects from your DB2 environment, and use this information to generate repository members documenting the imported DB2 objects.
- Chapter 8, "Commands," provides an alphabetical list of descriptions of the Manager Products commands that support your DB2 environment.
- Chapter 9, "Repository Member Types," supplies an alphabetical list of descriptions of the repository member types that you use to document DB2 objects.

Publication Conventions

These conventions apply to syntax diagrams that appear in this publication.

Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

Convention	Represents
➤➤	at the beginning of a line indicates the start of a statement.
➤	at the end of a line indicates the end of a statement.
——➤	at the end of a line indicates that the statement continues on the line below.
➤——	at the beginning of a line indicates that the statement continues from the line above.

Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

Statement identifiers appear on the main path of the diagram:

```
➤———— COMMAND ————————————————————————————➤
```

A required keyword appears on the main path:

```
➤———— COMMAND —— KEYWORD ————————————————➤
```

An optional keyword appears below the main path:

```
➤———— COMMAND ————————————————————————————➤
              └── KEYWORD ──┘
```

Convention	Represents

Where there is a choice of required keywords, the keywords appear in a vertical list; one of them is on the main path:

```
►──────COMMAND──────┬─KEYWORD1─┬──────────────────────►
                    ├─KEYWORD2─┤
                    └─KEYWORD3─┘
```

or

```
►──────COMMAND──────┬─KEYWORD1─┬──────────────────────►
                    ├─KEYWORD2─┤
                    └─KEYWORD3─┘
```

Where there is a choice of optional keywords, the keywords appear in a vertical list, below the main path:

```
►──────COMMAND──────┬──────────┬──────────────────────►
                    ├─KEYWORD1─┤
                    └─KEYWORD2─┘
```

The repeat symbol, <<<<<<, above a keyword or variable, or above a whole clause, indicates that the keyword, variable, or clause may be specified more than once:

```
                         <<<<<<<<
►──────COMMAND ────── KEYWORD──────────────────────────►
```

A repeat symbol broken by a comma indicates that if the keyword, variable, or clause is specified more than once, a comma must separate each instance of the keyword, variable, or clause:

```
                         <<<,<<
►──────COMMAND ────── KEYWORD──────────────────────────►
```

The repeat symbol above a list of keywords (one of which appears on the main path) indicates that any one or more of the keywords may be specified; at least one must be specified:

```
                                <<<<<<<<<<<<<<<<
►──────COMMAND──────┬─KEYWORD1─┬──────────────────────►
                    └─KEYWORD2─┘
```

The repeat symbol above a list of keywords (all of which are below the main path) indicates that any one or more of the keywords maybe specified, but they are all optional:

```
                                <<<<<<<<<<<<<<<<
►──────COMMAND──────┬──────────┬──────────────────────►
                    ├─KEYWORD1─┤
                    └─KEYWORD2─┘
```

ASG uses these conventions in technical publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.
Vertical Separator Bar (|) with underline	Options available with the default value underlined (e.g., Y|<u>N</u>).

ASG Customer Support

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours.

ASG Third-party Support. ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Intelligent Support Portal (ISP)

Online product support is available at: http://www.asg.com/support/support.asp via the ASG Intelligent Support Portal (ISP). Your logon information for ISP online support is:

Customer ID = *NNNNNNNNN*
Password = *XXXXXXXXXX*

where:

NNNNNNNNNN is your customer ID supplied by ASG Product Distribution.
XXXXXXXXXX is your unique password supplied by ASG Product Distribution.

The *ASG-Intelligent Support Portal User's Guide* provides instructions on how to use the ISP and is located on the ASG Support web page.

Telephone Support

To expedite response time, please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely as displayed
- A description of the specific steps that immediately preceded the problem
- Verify whether you received an ASG Service Pack or cumulative service tape for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack. You can access the latest software corrections and Service Packs via the ISP.
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578 or 800.775.5675	703.464.4901	support@asg.com
Australia	00.800 3544 3578 or 61.3.9645.8500	61.2.9460.0280	support.au@asg.com
England	00.800 3544 3578 or 44.1727.736305	44.1727.812018	support.uk@asg.com
France	00.800 3544 3578 or 33.141.028590	33.141.028589	support.fr@asg.com
Germany	00.800 3544 3578 or 49.89.45716.200	49.89.45716.400	support.de@asg.com
Italy	00.800 3544 3578 or 39.0290.4500.25		support.it@asg.com
Singapore	00.800 3544 3578 or 65.332.2922	65.337.7228	support.sg@asg.com
South Africa	00.800 3544 3578 or 00.800.201.423		support.sa@asg.com
All other countries:	1.239.435.2201		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578	**Netherlands**	00.800 3354 3578
Asia	00.800 3544 3578	**New Zealand**	00.800 3354 3578
Australia	00.800.3354 3578	**Singapore**	00.800 3354 3578
Denmark	00.800 3544 3578	**South Korea**	00.800 3354 3578
France	00.800.3354 3578	**Sweden**	00.800 3354 3578
Germany	00.800.3354 3578	**Switzerland**	00.800 3354 3578
Hong Kong	00.800 3544 3578	**Thailand**	00.800 3354 3578
Ireland	00.800 3544 3578	**United Kingdom**	00.800 3354 3578
Israel	00.800 3544 3578	**All other countries**	1.239.435.2201 or 1.602.667.2800
Japan	00.800 3544 3578		

If you receive a voice mail message, follow the instructions to report a production-down or critical problem. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible. Please have available the information described previously when the ASG Support representative contacts you.

ASG Documentation/Product Enhancement Fax Form

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/asp/emailproductsuggestions.asp.

If you do not have access to the web, FAX your suggestions to product management at (239) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Product Version # (*required*)	Publication Date
Product:		
Publication # (*required*):		
Tape VOLSER:		

Enhancement Request:

1 Introduction

This chapter includes these sections:

Features: The Tools Provided 2
Corporate Repository 3
Diagramming Functions 4
Data Modeling and Design Functions 4
Data Definition Language (DDL) Generator 5
COBOL, PL/I, and Assembler Language Generator 5
Dynamically Submitting SQL Statements to DB2 or SQL/DS 6
Importing Information from DB2 6

Functions: How to Use the Tools Provided 7
Standards 7
DB2 Database Design 8
Implementation 11
Maintenance 13
Summary 16

Benefits 17
A Shared and Reusable Corporate Model 17
Automated Design 19
Conclusion 20

ASG designs and builds Computer Aided Software Engineering (CASE) tools. These are state-of-the-art tools, designed to help business users solve problems with the development and maintenance of information systems.

Manager Products automates the major tasks involved in the design and implementation of DB2 and SQL/DS databases and the applications that use them: progress from pictures (analyst diagrams) through to practical solutions (databases and application programs).

Build and maintain an efficient DB2 or SQL/DS environment far faster than manual methods using our tools.

The tools are based upon using a corporate repository to store all of the information an organization needs to support business modeling, data processing, and application development. The tools share a common user interface (including online documentation and Help) which can be tailored to suit the environment used to and to suit users with different skills.

The import functions populate the corporate repository with members generated from information imported from DB2 and SQL/DS.

An understanding of DB2 and SQL/DS and their terminology is assumed.

Features: The Tools Provided

These tools and functions to support DB2 and SQL/DS system life cycles are provided:

- A corporate repository
- Diagramming functions
- Data modeling and design functions
- A Data Definition Language (DDL) generator
- A COBOL, PL/1, and Assembler language data structure generator
- Dynamic SQL functions
- Import to DB2 functions.

ASG recommends using all of the above to support the entire system life cycle. However, each tool can be used independently, as suits the purpose.

Corporate Repository

These are the features of the corporate repository:

- A repository in which to store information that supports business modelling data processing and application development activities
- Supports documentation of all DB2 and SQL/DS components including security information and DB2 plans
- Supports referential integrity
- Supports the documentation of data held in major DBMS and sharing of the documentation. For example, data shared between DB2 and IMS maybe documented once only and maintained centrally
- Supports documentation and control of historical, production, and development versions of a system and its components
- Populated using the following methods:
  - Using diagrams created on your programmable workstation (PWS)
  - Automatically, as the result of using data modeling and design functions
  - Directly, using definition statements
  - Importing information from external sources
- Interrogation and reporting capabilities.

Refer to Appendix B, "Documenting Other Relational Databases," on page 515 for details of sharing the documentation of different relational databases.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of using a repository.

Diagramming Functions

These are the features of the diagramming tool:

- Easy drawing of structured design diagrams including, for example, entity models and organizational charts on a programmable workstation
- Support for diagramming and generating entity relationship models
- Load diagrams to a repository on the host (mainframe) system
- Diagram validation for consistency and logic according to predefined rules
- A local repository to download a subset of the host repository
- Lock information in the host repository while it is on the programmable workstation
- Interrogate the host repository while creating diagrams
- Tailor diagram objects, connectors, menus, and validation rules to suit your environment
- Repository diagram generation produces hierarchical and network diagrams from definitions in the local repository

Data Modeling and Design Functions

These are the features of data modeling and design functions:

- An automated database design and information modeling tool
- Supports most data analysis methodologies, both top down business-oriented (entity modeling) and bottom-up application-oriented (userview modeling) approaches
- Reconciles top-down and bottom-up design information
- Produces a normalized logical database design (1st, 2nd, or 3rd normal form)
- Produces design reports that enable you to evaluate the design
- Automatically populates the repository with a first-cut physical design made up of tables, views, and indexes.

Refer to the *ASG-DesignManager User's Guide* for details of data modeling and design functions.

Data Definition Language (DDL) Generator

These are the features of the DDL generator:

- Generates DDL statements for the major DBMS such as DB2, SQL/DS, and IMS
- Generates SQL statements (CREATE, ALTER, DROP, GRANT, etc.) from repository documentation for submission to your DB2 or SQL/DS environment
- Supports referential integrity
- Generates impact analysis reports for DROP statements
- Uses a name reduction process to ensure that generated names are not too long for SQL
- Can be tailored to suit your installation's conventions and standards.

Refer to Chapter 5, "Export to DB2," on page 105 for details of DDL generation.

COBOL, PL/I, and Assembler Language Generator

These are the features of the source language generator:

- Generates host language data structures for inclusion in programs that access the DBMS
- Uses a name reduction process to ensure that generated names conform to the rules applying to the program language in which they are to be used
- Can be tailored to suit your installations conventions and standards
- Generates table layouts documenting the columns, edit procedures, field procedures, and validation procedures of tables and views.

Refer to Chapter 5, "Export to DB2," on page 105 for details of Assembler language, COBOL, and PL/I generation.

Dynamically Submitting SQL Statements to DB2 or SQL/DS

These are the features of dynamic SQL functions:

- Submits from within Manager Products to your relational environment, any SQL statement that can be dynamically prepared for execution
- Receives from your relational environment and within Manager Products, result tables generated in response to SQL queries
- Receives from your relational environment and within Manager Products, SQLCODEs and SQL/DS HELP text
- Creates executive routines containing embedded SQL statements that can:
  - Submit any SQL statements that can be dynamically prepared for execution
  - Create and populate a table
  - Insert rows into a table
  - Import information from your relational environment and assign it to Procedures Language variables.

Refer to Chapter 6, "Dynamic Import/Export," on page 131 for details of dynamic SQL functions.

Importing Information from DB2

These are the features of the import facilities:

- Imports information about DB2 or SQL/DS objects onto the Workbench Translation Area (WBTA)
- Generates proposed members from the imported information and provides:
  - A reconciliation report comparing the proposed members with existing repository members having the same name
  - The ability to change the proposed members. Commands are provided which enable you to work from the reconciliation report to make the changes.
  - The ability to tailor how proposed members are generated so that they suit your repository standards.
- Generates member definition statements for the proposed members in layouts which may be tailored to suit your repository standards
- Enters the proposed members into the repository.

Refer to Chapter 7, "Importing From DB2," on page 149 for details of importing information about DB2 objects.

Functions: How to Use the Tools Provided

Each of the functions provided supports part of a DB2 or SQL/DS system life cycle. The life cycle consists of these phases: design, implementation, and maintenance.

However, standards, particularly naming standards, need to be established and supported throughout the life cycle. Before using Manager Products consider how they enable you to promote standards throughout the life cycle.

Standards

Manager Products implement and control standards throughout the life cycle. Standards may include naming, database development, and application development standards.

If your systems are documented in the repository, it can help you and your development teams to identify homonyms (same name different thing) and can be checked for synonyms (same thing different name). Checking for the existence of synonyms is an easy matter if you have naming standards.

You can record several alternative names (aliases) for each definition in the repository, to suit different application environments. When you come to generate SQL statements, COBOL, PL/I, and/or Assembler language source direct from the repository, you can use the relevant alias name instead of the unique name, by which the definition is retrieved. A name reduction process is provided to ensure that the names generated are not too long for the target environment.

So, you can help your development teams to adhere to your standards by providing, via the repository, facilities that enable you to check and enforce them. If you do not have naming standards, getting them off the ground will be easier if your environment is controlled by use of a central repository.

In addition to naming standards you can promote your database and application development standards. You can tailor our generation tools so that the output they produce is automatically consistent with your standards.

Storing documentation of your standards in the repository or in the user definable help system that is provided ensures that they are centrally available.

Using Manager Products to control the system life cycle ensures that the systems built conforms to standards. This results in systems which are easier to understand and more effective communications.

DB2 Database Design

DB2 and SQL/DS database design involves two stages:

- Identification of the data and functions required to support a particular application or several interrelated applications, and determine how that data is to be stored
- Deciding on the operational aspects of the database, considering for example, the physical storage and performance requirements

Take referential integrity into account in both stages.

Identifying Data and Functional Requirements

Identify the data to be used in the database and define it in these forms:

Entity models. Data models that describe entities, their attributes, and the relationships between them and

Userview models. Definitions of the databases' end-user's requirements.

Use the diagram editor to draw entity models on a programmable workstation and then upload them to the host repository where they are converted into and held as definitions. Definitions in the repository can be interrogated, reported, and used by data modeling and design functions. The diagram editor checks diagrams, according to predefined rules, for validity, consistency, and completeness.

The host repository is always available while you are using diagramming functions. You can interrogate and report from the repository to obtain the information you need while you are creating your entity relationship diagrams. If you are updating information which is already in the repository, you can lock that information in order to prevent other users from updating it until you are finished.

The information recorded in userviews is the result of detailed investigations by the data analyst into existing documentation, table layouts, reports, and the requirements of the databases' end users. Userviews are entered as definitions directly to the repository.

Userview modeling is made much easier when your application systems are centrally documented in the repository and you have implemented naming standards. You can look up and document data element definitions and control names.

If you have used data modeling and design functions to design other DBMS databases and that work was based on an entity model that is still current, you can reuse that entity model and associated data element documentation already in your repository to design DB2 and SQL/DS databases.

For example, if you have recently designed an IMS database to support a transaction application and want to extract data to use in an end-user inquiry service based on a DB2 database, you can exploit the entity model you created for the IMS design to design the DB2 database. (This is one aspect of how our tools promote data sharing.)

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of the design process.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of adding definitions to the repository.

Logical Database Design

Once the entity models and userviews are documented in the repository, they can be moved onto the Workbench Design Area (WBDA) where you can design, automatically, a third normal form relational schema (logical database design). If particular circumstances such as stringent performance requirements dictate it, you can choose to normalize to first or second normal form only.

A wide variety of design reports enable you to identify problems such as missing entities, homonyms and synonyms, redundant and/or inconsistent end-user requirements, and thus to resolve any conflicts between your userviews (bottom-up view) and Entities (top-down view). The reports also enable you to analyze the generated referential structures.

Using these reports you can reline the design by adjusting your entity and userview models and iterating design on the WBDA until you are satisfied with the design and the you understand its referential structures.

Entity models are easily adjusted by redrawing entity relationship diagrams and (re-)uploading them to the host repository. Userview models are adjusted by updating the information in the repository, direct.

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of the design process.

Refer to the *ASG-ManagerView User's Guide* for details of creating entity model diagrams.

Physical Database Design

Once you are satisfied with the relational schema, it can be automatically converted to a first-cut physical design. The design comprises DB2 or SQL/DS tables, together with:

- Their respective primary keys and foreign keys and
- Unique indexes on the primary key for referential integrity.

You can also generate views, if they are required.

You can and should review the design, on the WBDA, to ensure that it meets your requirements. (At this point the design is represented as definitions of tables, indexes, and views.)

Once you are satisfied with the design on the WBDA, the repository can be automatically populated with definitions of the tables, views, and indexes that comprise the first-cut physical design. The table definitions include clauses for primary and foreign keys. You then complete the physical design in the repository.

Use the design reports to assist you to:

- Derive the referential structures of the tables
- Determine the DB2 table space or SQL/DS dbspace usage of related tables
- Decide the referential integrity constraints on the delete rules for tables

You can use the repository interrogation and reporting features to help generate the information you need to make the final design decisions regarding operational and performance requirements. For example, you can interrogate the repository to check which tables are stored in which DB2 storage groups. You can also estimate the storage space that will be required by tables and indexes in DB2.

Then you can add to the definitions comprising the design, information specific to operational and performance requirements.

Complete the physical design with these steps:

- Document the other object types required: SQL/DS dbspaces or DB2 databases, storage groups and table spaces as well as additional (non-primary) indexes.
- Document security and authorization information consisting of privileges and users.

Some of the useful features that support complete documentation of the design are:

- Data definitions in the repository use the same terminology and keywords as SQL.
- As with DB2's LIKE clause, you can specify that a table definition is to contain the same columns (and other characteristics) as an existing table definition.
- Columns are documented as separate definitions. This gives you greater control over data redundancy and data sharing between tables and between tables and other non DB2 or SQL/DS systems
- Since columns in different tables can share the same data elements and the same data elements can be shared by other DBMS, each data element definition can be associated with several table definitions and with repository documentation of other DBMSs such as IMS. (This is one aspect of how our tools encourage and help to control data sharing.)
- You can attach labels and comments to tables and views and to individual columns within them.
- You can define synonyms for users and generate SQL CREATE SYNONYM statements (very useful after dropping a database or dbspace and having to recreate large numbers of synonyms).

Refer to Chapter 9, "Repository Member Types," on page 341 for details of DB2 repository definitions.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of dictionary management commands.

Implementation

Exporting Your Design

When your database is fully documented in the repository, you can automatically generate all of the SQL statements required to define objects in the DB2 or SQL/DS Catalog.

You can also use the generation features to impose standards by tailoring them so that the output they produce conforms to your standards and procedures for application development.

Using the generators we provide, you can develop and maintain several DBMS applications by exporting the required database objects and program code from the same central data model.

Dynamic SQL functions enable you to submit SQL statements to DB2 or SQL/DS from within the Manager Products environment.

Refer to Chapter 5, "Export to DB2," on page 105 for details of SQL generation.

Refer to Chapter 6, "Dynamic Import/Export," on page 131 for details of dynamic SQL functions.

Developing Applications

Having implemented the design you will already have started work on building application programs that will access the data in the database.

You can generate the data declaration statements required in application programs, automatically, from the definitions of tables and views in the repository. Data declaration statements define the DB2 tables and views that the application accesses. You can use repository definitions of plans to generate BIND or REBIND subcommands, which produce application plans in DB2.

And you can generate data structures (that is, the host variables used to contain data transferred to and from DB2 and SQL/DS) for COBOL, PL/I, and Assembler language application programs.

Implementation will be smoother if you have used the repository to help you impose naming standards.

When you come to generate COBOL, PL/I, and/or Assembler language source, you can use the relevant alias name instead of the unique name which the definition is retrieved. We have also provided a name reduction process to ensure that the names you generate are not too long for the target environment.

By documenting your application programs in the repository, the impact of changes to tables and views can be measured easily by interrogating the repository. Thus the repository becomes an intelligent and active part of your change-control procedures.

Refer to Chapter 5, "Export to DB2," on page 105 for details of Assembler language, COBOL, and PL/I generation.

Monitoring and Tuning the Design

Following the initial (probably pilot) implementation of the database design you will want to experiment with the operational and performance aspects of the database design in order to improve the performance of the applications that use it.

This involves:

- Monitoring the database in order to identify if tuning is required and what needs to be done
- Adjusting the design in the repository, and then
- Regenerating the necessary components from the repository.

For example, un-normalizing certain tables for performance reasons may involve combining tables. You can interrogate the repository and generate SQL DROP statements with full impact analysis reports to find out exactly what the effect of dropping an object will be.

Of course the process of monitoring and tuning the database will not only follow the initial implementation; it will be continuous.

Maintenance

After the implementation has gone live, it will be necessary to maintain it. Activities typical during the maintenance phase are:

- Maintain the documentation of the implemented system and ensure that changes and enhancements are documented in the repository
- Interrogation; for example, "where else is this field used?"
- Reporting; for example, you may want to send a complete list of all your tables and views containing sales data to another office
- Adding columns to a table or changing some of its characteristics; for example, adding a foreign key
- Authorize new users to use certain tables
- Add or amend comments to tables and views
- Add synonyms to tables and views
- Maintain application programs.

Change Management in the System Life Cycle

Once the database and associated application development has gone live and the documentation in the repository is complete, it is likely that you will have to support ongoing development as well as production systems.

Use the status functions to maintain both production and development versions of the documentation in the repository.

Using statuses your development teams can create an updated version of the production system documentation without changing the production version and without duplicating what they do not change. Thus they can design, document and generate SQL, COBOL, PL/I, and Assembler language for a new or changed version of the system under development without affecting the production system and without being isolated from it.

Refer to the *ASG-Manager Products Status Concepts* manual for details about statuses.

Interrogation and Reporting

You can interrogate, report from, and produce documentation of table layouts from the repository in order to obtain information about the systems documented in it. For example, you can interrogate the relationships between definitions for the purpose of impact analysis and you can produce table layouts documenting tables and views and their columns, edit procedures, field procedures, and validation procedures.

Refer to "DB2 PRODUCE" on page 273 for details of table layouts.

You can index repository definitions using keywords and classifications that are meaningful in your environment. You can also add notes and other descriptive information about the system. All such information can be retrieved easily when it is required.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of dictionary management commands.

Using Dynamic SQL functions you can also interrogate your DB2 or SQL/DS environment from within Manager Products.

Refer to Chapter 6, "Dynamic Import/Export," on page 131 for details of Dynamic SQL functions.

Dropping and Altering Objects

Dropping objects is a powerful DB2 and SQL/DS feature: so powerful that it should be used with care. If you generate your SQL DROP statements from the repository you get a complete impact analysis report of what you will lose or affect before you drop the object.

An object may only need to be dropped temporarily in order for you to perform some major restructuring or maintenance: if you have a definition in the repository recreation of the object is very simply done.

You can also generate ALTER TABLE statements from the definitions of tables in the repository. This feature is task-driven; if it is necessary in order to achieve the alteration you want, several ALTER statements will be generated automatically.

Security and Authorizations

When you want to grant or revoke privileges to and from users you can automatically generate GRANT and REVOKE statements from the repository.

This is particularly useful if you have dropped a database (and everything that's inside it); you will usually need to re-GRANT many authorizations when the objects are recreated.

Application Maintenance

When changing application programs, it is often necessary to make careful trade-offs and technical decisions that require precise answers to queries of what other applications are affected.

You can obtain the information you need to make such decisions from the repository. For example, you can find out which programs use a particular column in a particular table if that column is going to change in some way.

When application programs need to change you can change the repository definition and regenerate data structures and DB2 data declaration statements easily.

Dynamic SQL Functions

Dynamic SQL functions enable you to dynamically submit SQL statements to DB2 or SQL/DS from within Manager Products, and to receive the results.

Any SQL statement that can be dynamically prepared for execution can be submitted.

You can submit SQL statements that have been previously generated from the repository by our Data Definition Language generator.

SQLCODEs and SQL/DS HELP text is displayed in response to unsuccessful statements.

You can interrogate your relational environment by submitting SQL SELECT statements and receiving the result tables the statements generate.

Implementation and maintenance of your DB2 or SQL/DS environment can therefore be carried out in the minimum amount of time.

For example, you could interrogate a table in DB2 or SQL/DS prior to using our DDL generator to generate an SQL ALTER statement, and then submit that statement using Dynamic SQL functions.

Refer to Chapter 6, "Dynamic Import/Export," on page 131 for details of dynamic SQL functions.

Import Functions

Our import facilities enable you to import information about DB2 or SQL/DS objects onto the Workbench Translation Area (WBTA) and to use that information to populate the corporate repository. These are the major benefits of the import facilities:

- Manager Products users who have not documented their DB2 or SQL/DS environment in the repository can do so in the minimum amount of time
- Users who *have* documented their environment can ensure that their existing documentation is complete and accurate by reconciling it with information imported from DB2 or SQL/DS.

Having documented your DB2 or SQL/DS environment in the repository, you can use Manager Products's CASE tools to analyze, maintain, and improve that environment.

Powerful repository management commands enable you to alter members generated from imported information so that they reflect any maintenance you intend to carry out in DB2 or SQL/DS.

Diagramming functions and data modeling and design functions enable you to analyze the repository in order to produce a normalized logical design from which the repository can be populated with a first-cut physical design. The import facilities therefore provide the first step in the re-engineering process.

Using our generators for Data Definition Language and for source languages you can generate SQL, COBOL, PL/I, and Assembler language from repository members.

Dynamic SQL functions enable you to dynamically submit generated SQL statements to DB2 or SQL/DS from within Manager Products, and receive the results.

Generated COBOL, PL/I, and Assembler language can be transferred to an external file for inclusion in your application programs.

Refer to Chapter 7, "Importing From DB2," on page 149 for details of the import facilities.

Summary

The system life cycle and how you can use the tools we provide to support your use of DB2 and SQL/DS can be summarized as follows:

- The design phase involves:
  - Building entity (top down analysis) and/or userview (bottom-up analysis) models
  - Generating, automatically, a logical design (3NF relational schema)
  - Generating a first-cut physical design in the repository
- The physical design is completed and documented in the repository and implementation involves exporting SQL statements generated from it to your DB2 or SQL/DS environment
- The maintenance phase involves:
  - Interrogating and reporting from the repository in order to obtain information about the system
  - Generating ALTER statements from the repository to reflect changing requirements for tables
  - Supporting security and authorizations in the database

Benefits

The corporate repository is a central repository of reusable information about the corporation, its data, and systems. The tools that we provide are productivity tools for database design and application development.

The benefits that you can gain from use of the corporate repository and the productivity tools based upon it, are described in this section.

A Shared and Reusable Corporate Model

The data used by most organizations changes very little in relation to the rate of change of the application systems.

Using the corporate repository, you can create a central data model in which each data element used by the corporation is defined only once, consistently, independent of any particular DBMS/application.

Using import functions you can populate the corporate repository in the minimum amount of time.

Using the generators we provide, you can develop and maintain several DBMS applications, automatically, by exporting the required database objects and program code from the central data model.

For example, if you use an IMS database for applications with high transaction rates and extract data from the IMS database to DB2, the repository helps you control copy management.

The central data model is the point of integration, control, and central reference shared between:

- Your development teams and the end users and management
- Design, generation, and other productivity tools.

A corporate model is created when you document additional objects related to the business of the enterprise, such as:

- Business plans, requirements, and rules
- Organizational units; for example, branches and departments.

So, the corporate repository becomes a communication tool to help you solve the communications problems between DP personnel, end users, and management; it provides a single source of all information across the full spectrum of your organization's applications. The fact that it is used actively in automated and semi-automated process throughout the life cycle ensures that its contents are reliable.

When combined with the use of a sound naming strategy, use of a corporate repository will enable your organization to achieve a common conceptual view of your information resource and vastly improve understanding and communications about that system.

The benefits include:

- Accurate and up-to-date documentation is readily available in one safe place
- Centralized information about your organization and its systems: multiple computer systems, locations, DBMSs, databases, etc.
- Fast and accurate impact analysis at a corporate and local level. For example, you can find out what the impact of a proposed change will be on your:
  - Existing systems: reports, programs, databases
  - Developing systems
  - Organizational units.
- A data model that is independent of the physical implementation. You can tune the physical design implementation without losing the logical (and therefore, theoretically the best) design, from which it was derived.
- A data model that can be reused for design work across applications and life cycles
- Elimination of data duplication across applications because the repository helps developers to find out what data already exists
- Data sharing that can be promoted and controlled
- Normalized data structures that minimize redundancy and maximize flexibility to future change
- Standards can be introduced into the system development life cycle. For example, the repository helps you to establish and foster meaningful naming standards and, because the data model is driving all of the application Systems, throughout the organization.
- Design information such as entity relationship diagrams and entity models can be documented and is readily available for reuse.

All of the above reduce the time, resources, confusion, and expense involved in building the information systems required to satisfy your organization's information needs. They help end users, DP personnel, and management to understand each other and do a better job.

Automated Design

A poorly designed database will result in time consuming and expensive software revisions when business requirements change.

The benefits of creating a logical database design that is independent of the chosen DBMS, has become apparent as the use of databases and the associated DBMSs has increased and information systems become larger and more complex.

By providing a detailed and clear model of the systems information requirements, a logical database design improves communications between analysts and end-users. It ensures that the fundamental data structures required to support the end-users' information needs are identified. It enables you to understand the whole business, not just one application.

The value of a logical database design depends on several factors:

- Whether it describes all the types of data (data elements) that will be required in the database
- Whether it accommodates all the end users' requirements of the database (the userviews)
- Whether, when implemented physically, it minimizes the duplication of the data values that will be included in the database
- How well it can accommodate new requirements of the database, and the addition of further types of data, as the database and associated applications evolve.

The corporate repository provides storage, documentation, and cross-referencing facilities for the definition of data elements and userviews.

The amalgamation, analysis, and structuring of all the data elements and userviews to produce a logical database design model that satisfies the requirements for:

- Minimal duplication of data values in the database
- Optimal stability with database evolution
- Referential integrity

is all done automatically by data modeling and design functions. If done manually, these tasks are time-consuming, complex, and error-prone.

Using data modeling and design functions, database designers are free to devote themselves to aspects of the design process that require human judgment, such as:

- Evaluation of the logical model and userviews for omissions and errors
- The physical implementation of the model in the database
- Weighing the advantages and disadvantages of adjusting the physical model to match the available hardware and meet particular performance requirements.

So, your database designers can:

- Build and verify logical and physical models of current and potential information delivery systems
- Design and implement optimized database structures reflecting current and potential needs
- Exploit and achieve maximum benefit from relational technology.

All of the above reduce the time and expense involved in design and implementation of the systems required to satisfy your organizations information needs. The databases produced in such an environment are well designed and therefore responsive to the need for database evolution.

Conclusion

Using our tools your organization can respond rapidly to change and reduce the cost of application systems design and implementation. Although your organization has to invest money and resources in the building-up and maintaining of the corporate repository, using it you can achieve:

- Consistency across systems
- Understanding of systems despite changing human resource and consequent improvements in maintenance and development
- Better quality and more reliable systems.

Through automation of the major parts of the development life cycle, you can progress from pictures on the programmable workstation to practical solutions (the actual database and the applications based upon it) relatively fast.

You are using one product set that exploits the best of personal computer and mainframe technology. You don’t have to become familiar with several different products from several different vendors.

2 What Do You Want To Do?

This chapter directs you to the documentation relevant to the task you wish to perform. For each task, the relevant section and page in this publication is given.

ASG Support for Your DB2 Environment

Topic	Page
Designing a DB2 Database	21
DB2 Object Definition	22
Generating Output	23
Dynamically Submitting SQL Statements	25
Importing from DB2	25

Designing a DB2 Database

DB2 Database Design

Topic	Page
Designing a DB2 Database	32
Producing Output Describing the DB2 Design	22
DB2 PREVIEW	262
DB2 POPULATE	250

Producing Output Describing the DB2 Design

Topic	Page
DB2 LIST TABLES	240
DB2 LIST CYCLES	239
DB2 REPORT	288
DB2 PLOT CLUSTER	242
DB2 PLOT REFERENTIAL-STRUCTURES	245

DB2 Repository Definition

Documenting a DB2 Dictionary Schema

Topic	Page
Documenting DB2 Objects	92
Documenting DB2 Security Information	96
Interrogating Your DB2 Dictionary Schema	102
Naming Guidelines	101

DB2 Object Definition

Topic	Page
DB2-ALIAS	342
DB2-COLLECTION	350
Documenting the Columns of Indexes, Tables, and Views	94
DB2-DATABASE	351
DB2-DMS	355
DB2-INDEX	370
DB2-LOCATION	390
DB2-PACKAGE	392
DB2-PLAN	398
DB2-PRIVILEGE	405

Topic	Page
DB2-PROCEDURE	417
DB2-STOGROUP	422
DB2-TABLE	425
DB2-TBSPACE	456
DB2-TRIGGER	472
DB2-USER	476
DB2-VIEW	481

Export to DB2

Generating Output

Topic	Page
DB2 CREATE	212
DB2 DROP	228
DB2 SYNONYM	295
DB2 ALTER	176
DB2 GRANT and DB2 REVOKE	234
DB2 COMMENT and DB2 LABEL	206
DB2 DECLARE	223
DB2 PRODUCE	273
Data Types Generated from Form Descriptions	522
DB2 BIND and DB2 REBIND	195
DB2 DEBUG	219
DB2 SIZE	291
DB2 RECALCULATE	282

Topic	Page
Sending Generated Output to a USER-MEMBER	334
Submitting Generated Output to Your Relational Environment	107
Tailoring Output	109

Tailoring Generated Output

Topic	Page
Generating Object Names and External Names from Aliases	114
Tailoring DATE and TIME Character Field Lengths	115
Generating an SQL DECLARE Statement with a Host Language Data Structure	117
Setting the Release of DB2	117
Setting the Release Flag	117
Generating Flat or Nested Data Structures	117
Generating Indicator Structures	118
Generating Indicator Suffixes on Structures	118
Setting Suffixes Applied to Indicator Array Names	119
Setting Suffixes Applied to Variable-Length Column Names	119
Automatically Generating SQL COMMENT ON/LABEL ON Statements	119
Generating One-, Two-, or Three-part Names for DB2 Objects	120
Setting a Tolerance Level for Output	121
Setting the SQL Escape Character	121
Setting Width of Output for SQL COMMENT ON Statements	122
Setting Width/Indent of the SQL DROP Impact Analysis Report	122
Allowing an Optional Space Character when Generating SQL DECIMAL Datatypes	122
Accessing a Specific DB2 Subsystem or Plan	123

Topic	Page
Setting EXPORT Generated Object-name Length	123
Setting the Generated Column Data Type	124
Taking User Exits when Accessing a Repository Member	125
Taking User Exits For Specified DB2 Export Functions	125
Taking User Exits for an Individual Export Function	129
Setting Switches for Non-standard SQL Generation	130

Dynamically Submitting SQL Statements

Topic	Page
Creating and Populating a Table	141
Inserting Rows into a Table	142
Importing Information and Assigning it to Command Variables	144
Submitting Any SQL Statement that Can Be Prepared	146
Creating Your Own HELP Text	147

Importing from DB2

Topic	Page
EXTRACT DB2	299
RADD	316
PREVIEW IMPORT	312
Tailoring Import	157

3 DB2 Database Design

This chapter includes these sections:

Introduction to DB2 Database Design 28
Support for Referential Integrity 30
Introduction to Referential Structures and Cycles 31
Features to Support DB2 31

Designing a DB2 Database 32
Creating Entity and Userview Models 32
Generating a Relational Schema 33
Generating the DB2 Design 34
Reporting the DB2 Design 35
Populating the Dictionary with DB2 Members 36
Examples of the DB2 Database Design Process 37

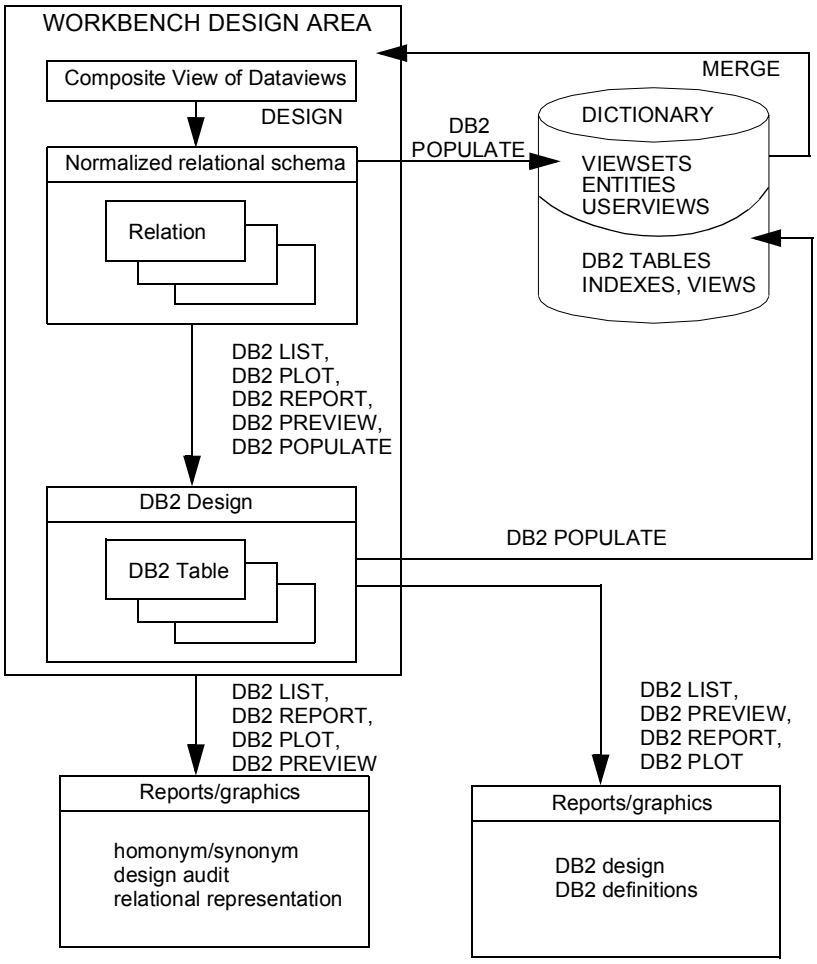
DB2 Command Output 54
Output from the DB2 REPORT Command 54
Output from the DB2 PLOT CLUSTER Command 63
Output from the DB2 PLOT REFERENTIAL-STRUCTURES Command 71
Output from the DB2 LIST TABLES Command 81
Output from the DB2 LIST CYCLES Command 83

Generated DB2 Member Definitions 85
Generated DB2-TABLE Member 85
Generated DB2-INDEX Member 86
Generated DB2-VIEW Member 87
Generated SYSTEM Member 88

Introduction to DB2 Database Design

Figure 1 represents the support provided for DB2 at the different stages of the DB2 database design process. From top to bottom, it shows the commands used at each stage to cause data to be input to the Workbench Design Area (WBDA), processed within it, or output from it. The contents of the dictionary and WBDA and the outputs from the DB2 commands are also shown.

Figure 1 • DB2 Database Design



There are two major stages of DB2 database design in which ASG-DesignManager participates:

- Identifying the data and functions required to support a particular application, or several interrelated applications, and determining how that data is to be stored
- Deciding on the operational aspects such as physical storage and performance requirements

DB2 provides data consistency among tables through referential constraints. The enforcement of referential constraints, called *referential integrity*, ensures that all references from one table to another are valid. Referential integrity spans both stages of DB2 database design. It involves:

- Determining the primary keys and foreign keys for tables
- Deciding the referential constraints on the delete rules for the tables
- Defining tablespaces containing referential structures.

You can automate much of the DB2 database design procedure as follows:

- Define your data in the dictionary as userview and entity models, containing USERVIEW and ENTITY members (collectively known as data-views)
- Create a composite view of the data in the WBDA by moving data in from the models, using the MERGE command
- Create a relational schema in the WBDA, by using the DESIGN command both to normalize the data to first, second, or third normal form, and to generate relations from the normalized data
- Evaluate the relational schema, using the ASG-DesignManager PLOT and REPORT commands
- Generate the DB2 design in the WBDA from the relational schema, using any of the DB2 LIST, DB2 PLOT, DB2 REPORT, DB2 PREVIEW, or DB2 POPULATE commands. The DB2 design consists of DB2 tables, with their primary and foreign keys, as well as table indexes and views.
- Evaluate the tables in the DB2 design, using output from the DB2 LIST, DB2 PLOT, and DB2 REPORT commands
- Use the output from the DB2 LIST CYCLES and PLOT REFERENTIAL-STRUCTURES commands to help plan which tablespaces to use for storing the tables and which referential constraints to be used with the delete rules for the tables

- Generate (in the WBDA) and inspect dictionary definitions of DB2-TABLE, DB2-INDEX, and/or DB2-VIEW members, using the DB2 PREVIEW command. In addition, a dictionary SYSTEM member can be defined containing a list of all the other generated dictionary members
- If the PREVIEWed definitions are satisfactory, then populate the dictionary with the definitions, using the DB2 POPULATE command
- Complete the physical design of the DB2 database by adding operational/performance information to the dictionary definitions.

This convention also applies to indexes and views.

Note:

If you wish, you can generate SQL CREATE TABLE statements straight from the WBDA by using the PRODUCE DB2 command after you have generated the relational schema. However, referential integrity is not supported by this command.

Support for Referential Integrity

DB2 provides data consistency among tables through referential constraints. The enforcement of referential constraints, called referential integrity (RI), ensures that all references from one table to another are valid. DB2 uses the primary keys and foreign keys in DB2 tables to enforce RI.

ASG-DesignManager supports RI throughout the DB2 database design process. The support includes:

- Determining the primary keys and foreign keys for the tables
- Helping you choose the referential constraints to be used with the delete rules for the tables
- Helping you define tablespaces containing referential structures.

DB2 primary and foreign keys are generated when the relations in the relational schema are converted to tables in the DB2 design. The foreign key relationships in which each table participates are also derived.

When you use the DB2 PREVIEW or the DB2 POPULATE command to generate DB2-TABLE dictionary definitions from the tables in the WBDA, they automatically include primary key keywords and foreign key clauses to support RI unless you specify the keyword NO-RI in the command to indicate that those clauses must be suppressed.

When relevant, a foreign key clause is generated for each foreign key relationship in which the table participates as a dependent table. It includes each of the columns comprising the foreign key. If the relationship is of the domain type, each column in the foreign key is matched with the corresponding column in the primary key of the parent table. The name of the parent table is included in each foreign key clause.

The output from the DB2 PLOT and DB2 REPORT commands shows the relationships which each selected table has with other tables in the WBDA. You can use the output from these commands to help you identify the referential structures of the tables, and to plan the tablespaces in which the DB2 tables should be stored.

The report outputs can help you design tile referential constraints for the delete rules which apply to the DB2 tables, but before you can specify delete rules you need to know the referential structure to which a table belongs and to be aware of any cycles.

Introduction to Referential Structures and Cycles

A referential structure may be described as a set of tables and relationships in which each table is a parent or dependent of itself or of another table in the set. Each table that is a parent or dependent is part of exactly one referential structure.

A cycle may be described as a path of relationships which connects a table to itself, in which the arrows representing the relationships all flow in the same direction. You have identified a cycle if you find the same table twice while tracing the dependent tables in a referential structure. The presence of a cycle in a referential structure affects the specification of delete rules, since a table must not be delete-connected to itself.

Awareness of referential structures and cycles is vitally important when making your final decisions about the delete rules to apply to particular tables and about the tablespaces in which the tables are to be stored.

The output of the DB2 PLOT REFERENTIAL-STRUCTURES command displays the referential structures present in the DB2 design generated in the WBDA. The DB2 LIST CYCLES command lists all cycles and, for each, the tables comprising the cycle.

Once you have decided on delete rules, you can specify them in each DB2-TABLE member by updating the FOREIGN-KEY clause.

Features to Support DB2

The relational schema in the WBDA automatically converts to a DB2 design containing DB2 tables, indexes, and views when you enter any of these commands (DB2 referential integrity is fully supported):

DB2 LIST TABLES command. Produces a list of some or all of the tables in the DB2 design generated in the WBDA.

DB2 LIST CYCLES command. Produces a list of all the cycles found in the design and, for each, the tables comprising the cycle.

DB2 REPORT command. Shows, for each selected table in the WBDA, the columns comprising the table, the dependencies represented by the table, and any other tables to which it is related.

DB2 PLOT CLUSTER command. Shows, for each selected table in the WBDA, a diagram in cluster form of its relationships with other tables and a matrix of all the tables in the WBDA, showing all relationships which exist between them.

DB2 PLOT REFERENTIAL-STRUCTURES command. produces an overview plot of the referential structures in the WBDA.

Once the DB2 design has been generated, you can generate dictionary definitions for DB2 tables, indexes, and views.

DB2 PREVIEW command. Enables you to preview the generated DB2 dictionary definitions before populating the dictionary with them.

DB2 POPULATE command. Automatically populates the dictionary with the following member types (if you have the optional DB2 Definition facility installed):

- DB2-TABLE
- DB2-INDEX
- DB2-VIEW
- SYSTEM

PRODUCE DB2 command. Generates SQL CREATE TABLE statements directly from the relations in the relational schema.

Designing a DB2 Database

Creating Entity and Userview Models

Having identified the data and functions needed to support a particular application, or several inter-related applications, you begin the DB2 database design process by defining your data in the dictionary. The definitions are in the form of entity models and userview models.

Entity models. Data models composed of ENTITY members in which data elements are defined as the attributes of entities and relationships are defined between the entities.

Userview models. Data models composed of USERVIEW members in which dependencies between data elements are defined that satisfy the requirements of the database end users.

Entities and userviews are collectively known as dataviews. Avoid using homonyms and synonyms when naming data elements. USERVIEW and ENTITY members should refer to validly named data elements.

Refer to the *ASG-DesignManager User's Guide* for details of dataviews, entities, and userviews, and for details of naming data elements.

Generating a Relational Schema

After you have created entity and userview models in the dictionary, you can create a relational schema.

Use the MERGE command to move data from the dictionary models into the WBDA to build up a single composite view of the data (consisting of functional, multivalued, and domain dependencies).

Next, use the DESIGN command to:

- Normalize the dependencies to first, second, or third normal form (1NF, 2NF, or 3NF)
- Identify potential keys and generate the relations of the relational schema

How far you wish to normalize the data depends on the needs of your installation; you can generate a first-cut DB2 design from data in first, second, or third normal form.

The relational schema consists of a set of relations, each containing a key and usually some non-key (non-prime) data elements. Each relation is identified by a unique WBDA number.

Using the output of the PLOT and REPORT commands, you can evaluate the relational schema to ensure that it satisfies your data access requirements. If not, you can modify the input, re-MERGE and re-DESIGN the relational schema in iterative fashion until the schema does meet your requirements. If you have a large amount of data to be evaluated, it is easier to report it a bit at a time, so that you can analyze several reports separately, rather than analyze one large one.

Once you are satisfied with the relational schema, you can generate a DB2 design from it in the WBDA by issuing any ASG-DesignManager DB2 command. Each relation is mapped into a table which is assigned the name and WBDA number of the relation from which it is generated. It is important to make sure that every relation is named, because, subsequently, dictionary definitions will not be generated from unnamed tables.

You may use the PRODUCE DB2 command at this point to generate SQL CREATE TABLE statements straight from the WBDA.

Refer to the *ASG-DesignManager User's Guide* for details of MERGE, DESIGN, and the relational schema.

Generating the DB2 Design

The DB2 design comprises tables generated directly from the relations in the relational schema in the WBDA. It is stored in the WBDA along with the relational schema.

Each table is assigned the name and WBDA number of the relation from which it is generated. Usually, each data element in the relation maps into a column of the table with the same name. In addition, ASG-DesignManager identifies the primary key and any foreign keys in each table (from the corresponding keys of the relation) for referential integrity, as indicated in the correspondence table below, and also identifies the foreign key relationships in which the table participates.

You should issue the DB2 LIST, DB2 PLOT, or DB2 REPORT command to generate and report the DB2 design for the first time. Using the output from these commands, you can examine and evaluate the tables of the generated design. You should ensure that the contents of the tables and their relationships with each other satisfy your database access requirements before you start to generate dictionary definitions from the tables.

You can also generate the DB2 design by entering the DB2 PREVIEW or DB2 POPULATE commands, but these commands do not provide a detailed report of the tables in the DB2 design.

Table 1 shows the correspondence between the relations in the relational schema and the tables in the DB2 design.

Table 1 The Correspondence between the Relations in the Relational Schema and the Tables in the DB2 Design

Relational Schema	DB2 Design
Relation name, WBDA number	Table name and WBDA number
Data elements in relation	Columns in table
Set of one or more non-prime data elements in role relation (where a domain dependency holds from key of relation to the set non-prime data elements)	Dropped from table (because, in each row, the set of values would be identical to that of the set of data elements comprising the of primary key)
Primary key of relation	Primary key of table
Foreign key of relation (except in the case of a domain association; see below)	Foreign key of table
Foreign key, hierarchical-one, and domain associations	Foreign key relationships

The relational schema contains a number of different types of associations, but in DB2 there is only one type of foreign key relationship.

When comparing graphical displays of the relational schema and the DB2 design, you will notice that the direction of an association in the relational schema is opposite to that of the corresponding relationship in the DB2 design.

In a relational schema, all foreign key, hierarchical-one, and domain associations are directed from a source relation containing a foreign key to a target relation containing the corresponding primary key. That is, the primary key of the target relation is a non-key set of data elements in the source relation.

In DB2 a foreign key relationship is directed from a parent table containing a primary key to a dependent table containing a foreign key. The foreign key is the same set of data elements as the foreign key of the corresponding source relation except in the case of a domain association, where the primary key of the dependent table is itself the foreign key to the parent table.

Refer to "DB2 Command Output" on page 54 for details of the output from the DB2 LIST, DB2 PLOT, and DB2 REPORT commands.

Reporting the DB2 Design

Use the DB2 LIST, DB2 PLOT, and DB2 REPORT commands to produce listings, plots and reports of the tables in the DB2 design. Any of these commands will cause the relational schema to be converted to a DB2 design, if one does not exist already.

You should use the reports to check that the following meet your database access requirements:

- Table columns
- Table primary keys
- Table foreign keys
- Relationships between tables.

If you find that tables do not contain the columns or keys that you expected, or that tables are not related to each other in the way that you want them to be, you can:

- Modify the ENTITY and USERVIEW members in the dictionary
- Re-MERGE and re-DESIGN the data to produce a new relational schema
- Generate a new DB2 design.

You can repeat this process as often as you want until the results are satisfactory.

The outputs from these commands show how tables in the DB2 design are related to each other. Furthermore, they show the referential structures and cycles present in the design. You will need this information to plan the tablespaces in which the tables will be stored. You will also need it when you design the referential integrity constraints for the tables' delete rules, as you refine the DB2 design.

Refer to "DB2 Command Output" on page 54 for details of the output from the DB2 LIST, DB2 PLOT, and DB2 REPORT commands.

Populating the Dictionary with DB2 Members

When you are satisfied with the DB2 tables in the WBDA, you can begin generating DB2-TABLE, DB2-INDEX, and DB2-VIEW dictionary member definitions.

You can also generate a SYSTEM dictionary definition containing a list of the names of all the DB2 members generated by the same command.

By now you should be at a sufficiently advanced stage in your design process for all the tables in the WBDA to be named (that is, by use of the NAME command to name the relations in the relational schema). ASG-DesignManager will not generate dictionary definitions from unnamed tables.

You should first issue a DB2 PREVIEW command. This generates and reports DB2 dictionary definitions from the DB2 design in the WBDA without adding them to the dictionary, so that you can first check them. Like the earlier stages of the design process, previewing can be iterative -- you can keep generating a new DB2 design in the WBDA and previewing the dictionary definitions until you are satisfied with them.

When you are satisfied with the definitions, you then can use the DB2 POPULATE command to populate the dictionary with them. DB2 PREVIEW and DB2 POPULATE generate exactly the same definitions, so by previewing the definitions you already know what will go into the dictionary.

In the commands, you can specify that dictionary definitions are to be generated either from all the DB2 tables in the WBDA or from a selection of tables indicated by name or by WBDA number. You can also specify that the tables are to be processed in alphanumeric order of table name.

When you generate DB2-TABLE members, the PRIMARY-KEY keywords, and foreign key CONSTRAINT clauses needed to support referential integrity (RI) are generated automatically unless you specify the keyword NO-RI in the command to indicate that they are to be suppressed.

You can also start to assign tables to tablespaces at this stage. You may choose to have one tablespace per table, or to store a whole referential structure in one tablespace, or to have one or more referential structures spanning tablespaces.

The final decisions about how tables will be stored cannot be made until you have ascertained the referential structures and cycles to which individual tables belong.

This information is provided by output from the DB2 PLOT REFERENTIAL-STRUCTURES and DB2 LIST CYCLES commands.

With the DB2 PREVIEW or POPULATE command, you can generate a DB2-VIEW member for each DB2-TABLE generated so that users do not access the DB2 tables directly.

In addition, a DB2-INDEX member, representing a primary key index, can be generated for each selected table.

You can tailor the format of the generated dictionary definitions by specifying, in the DB2 PREVIEW or POPULATE command, the name of a predefined FORMAT member to be used for formatting, provided that the optional User Formatted Output facility is installed.

Refer to "DB2 Command Output" on page 54 for details of the output from the DB2 LIST, DB2 PLOT, and DB2 REPORT commands.

Refer to "DB2 PREVIEW" on page 262 for details of the DB2 PREVIEW command and "DB2 POPULATE" on page 250 for details of the DB2 POPULATE command.

Examples of the DB2 Database Design Process

Introduction to Examples

In these two examples, you are taken through the DB2 design process and the method of refining the design. Both examples show specifically how domain associations in the relational schema are converted into DB2 foreign key relationships. The examples are the Department model and the Parts model.

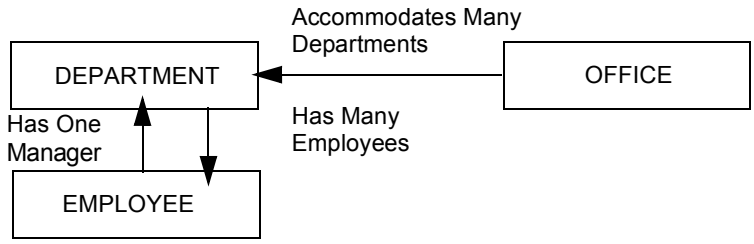
When displaying the relational schema in these examples, associations are directed from the source relation (containing the foreign key) to the target relation (containing the corresponding primary key). When displaying the DB2 design, relationships are directed from the parent table to the dependent table—the reverse direction.

Department Model Example

The Department model example shows how to define the relationships between employees (including managers), their departments, and the offices in which the departments are situated.

There are three entities in this organization; they represent employees, departments and offices, and their interrelationships, as shown in Figure 2.

Figure 2 • Departmental Model Example: Diagram Showing Entities and Their Interrelationships



Begin by defining the ENTITY members in the dictionary and naming them EMPLOYEE-ENT, MANAGER-ENT, DEPARTMENT-ENT, and OFFICE-ENT, as shown below.

```
EMPLOYEE-ENT
 ENTITY
 IDENTIFIER ISEMPLOYEE-NO
 ONE-ATTRIBUTES AREEMPLOYEE-NAME
 SUB-ENTITIES AREMANAGER-ENT

MANAGER-ENT
 ENTITY
 IDENTIFIER ISMANAGER-NO

DEPARTMENT-ENT
 ENTITY
 IDENTIFIER ISDEPARTMENT-NO
 ONE-ATTRIBUTES AREDEPARTMENT-NAME
 ONE-ASSOCIATION TOMANAGER-ENT
 MULTI-ASSOCIATION TOEMPLOYEE-ENT

OFFICE-ENT
 ENTITY
 IDENTIFIER ISOFFICE-LOCATION
 ONE-ATTRIBUTES AREOFFICE-NAME
 MULTI-ASSOCIATION TO DEPARTMENT-ENT
```

The entity EMPLOYEE-ENT represents an employee. It has:

- The identifier (key attribute), EMPLOYEE-NO
- A one-attribute (non-prime attribute), EMPLOYEE-NAME
- A sub-entity, MANAGER-ENT.

The sub-entity MANAGER-ENT also represents an employee, but one who plays the 'role' of a manager. It has the identifier, MANAGER-NO. The domain (or set of all valid values) of MANAGER-NO is a subdomain or subset of the domain of EMPLOYEE-NO, reflecting the fact that every manager is also an employee.

Thus, each employee is identified by an employee number, has a name, and may be a manager.

The entity DEPARTMENT-ENT represents a department. It has:

- The identifier (key attribute), DEPARTMENT-NO
- A one-attribute (non-prime attribute), DEPARTMENT-NAME
- A one-association (one-relationship) to the entity MANAGER-ENT
- A multi-association (many-relationship) to the entity EMPLOYEE-ENT.

That is, each department is identified by a department number, has a name and one manager, but may contain many employees.

The entity OFFICE-ENT represents an office location. It has:

- The identifier (key attribute), OFFICE-LOCATION
- A one-attribute (non-prime attribute), OFFICE-NAME
- A multi-association (many-relationship) to the entity DEPARTMENT-ENT.

That is, each office is identified by an office location, has a name, and can accommodate many departments.

If the entities EMPLOYEE-ENT, DEPARTMENT-ENT, and OFFICE-ENT are merged into the WBDA, using the MERGE command, the dependencies in Table 2 are derived:

Table 2 Department Model example: Derived Dependencies

Dependency			Derived from Entity
EMPLOYEE-NO	→	EMPLOYEE-NAME	EMPLOYEE-ENT
MANAGER-NO	⇢	EMPLOYEE-NO	EMPLOYEE-ENT
DEPARTMENT-NO	→	DEPARTMENT-NAME	DEPARTMENT-ENT
DEPARTMENT-NO	→	MANAGER-NO	DEPARTMENT-ENT
DEPARTMENT-NO	→→	EMPLOYEE-NO	OEPARTMENT-ENT
OFFICE-LOCATION	→	OFFICE-NAME	OFFICE-ENT
OFFICE-LOCATION	→→	DEPARTMENT-NO	OFFICE-ENT

You can now normalize the dependencies of the composite view, identify potential keys, and generate relations using the DESIGN command. The generated relations constitute the relational schema.

Refer to the *ASG-DesignManager User's Guide* for details of how dependencies are derived from ENTITY definitions.

The relational schema generated by the DESIGN command for the Department model example contains six relations representing the dependencies in the WBDA. The dependencies were generated from the entities, EMPLOYEE-ENT, MANAGER-ENT, DEPARTMENT-ENT, and OFFICE-ENT.

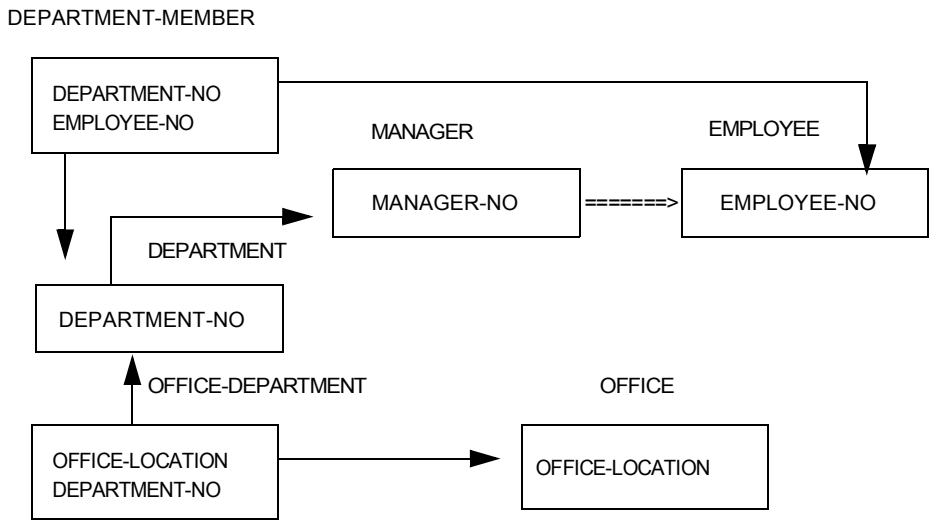
Table 3 shows the data elements in each relation. The names of data elements comprising the key of each relation are shown in capitals. The names of the non-prime data elements are shown in lower case.

Table 3 Table of Relations and their component Data Elements

Relation Name	Data Elements Comprising Relation		
EMPLOYEE	EMPLOYEE-NO	employee-name	
MANAGER	MANAGER-NO	employee-no	
DEPARTMENT	DEPARTMENT-NO	department-name	manager-no
DEPARTMENT-MEMBER	DEPARTMENT-NO	EMPLOYEE-NO	
OFFICE	OFFICE-LOCATION	office-name	
OFFICE-DEPARTMENT	OFFICE-LOCATION	DEPARTMENT-NO	

Figure 3 shows the six relations with their respective primary key data elements and the foreign key associations, domain associations, and hierarchical-one associations in which they participate.

Figure 3 • Department Model Example: Diagram Showing Associations of the Relations



The contents of the relations are described below, including, for each, its key, its non-prime data elements (if any) and any associations in which it participates as the source relation.

MANAGER is a role relation. It has:

- The key MANAGER-NO
- The non-prime data element employee-no
- A domain association leading from it to the target relation EMPLOYEE, via the domain dependency from its key MANAGER-NO to the key EMPLOYEE-NO of the target relation.

DEPARTMENT is an FD relation. It has:

- The key DEPARTMENT-NO
- The non-prime data elements department-name and manager-no
- A foreign key association leading from it to the target relation MANAGER, where its non-prime data element, manager-no, is also the key MANAGER-NO of MANAGER.

OFFICE-DEPARTMENT is an MVD (all-key) relation. It has:

- A composite key consisting of the prime data elements OFFICE-LOCATION and DEPARTMENT-NO
- A hierarchical-one association leading from it to the target relation DEPARTMENT. The prime data element DEPARTMENT-NO, of OFFICE-DEPARTMENT, is also the key of DEPARTMENT.
- A hierarchical-one association leading from it to the target relation OFFICE. The prime data element OFFICE-LOCATION, of OFFICE-DEPARTMENT, is also the key of OFFICE.

DEPARTMENT-MEMBER is an MVD (all-key) relation. It has:

- A composite primary key comprised of the data elements DEPARTMENT-NO and EMPLOYEE-NO
- A hierarchical-one association leading from it to the target relation DEPARTMENT. The prime data element DEPARTMENT-NO, of DEPARTMENT-MEMBER, is also the key of DEPARTMENT.
- A hierarchical-one association leading from it to the target relation EMPLOYEE. The prime data element EMPLOYEE-NO, of DEPARTMENT-MEMBER, is also the key of EMPLOYEE.

EMPLOYEE is an FD relation. It has:

- The key EMPLOYEE-NO
- The non-prime data element employee-name
- No associations leading from it.

OFFICE is an FD relation. It has:

- The key OFFICE-LOCATION
- The non-prime data element office-name
- No associations leading from it.

Refer to the *ASG-DesignManager User's Guide* for details of the structure of the relational schema.

You can now generate the DB2 design from the relational schema, by issuing any of these commands:

- DB2 LIST
- DB2 REPORT
- DB2 PLOT
- DB2 PREVIEW
- DB2 POPULATE.

The DB2 design generated from the relations EMPLOYEE, MANAGER, DEPARTMENT, DEPARTMENT-MEMBER, OFFICE, and OFFICE-DEPARTMENT contains six tables.

When a table in the DB2 design is generated from a relation in the relational schema, the table takes the name and WBDA number of its source relation. In general, all of the data elements of the relation become the columns of the table, where the columns that correspond to the key of the relation become the primary key of the table.

But in this example the role relation MANAGER is an exception. Its non-prime data element employee-no is omitted from the table because the set of valid values of MANAGER-NO is a subset (subdomain) of the set of valid values of EMPLOYEE-NO. If employee-no was included, each row of the table would contain the same value for both MANAGER-NO and employee-no.

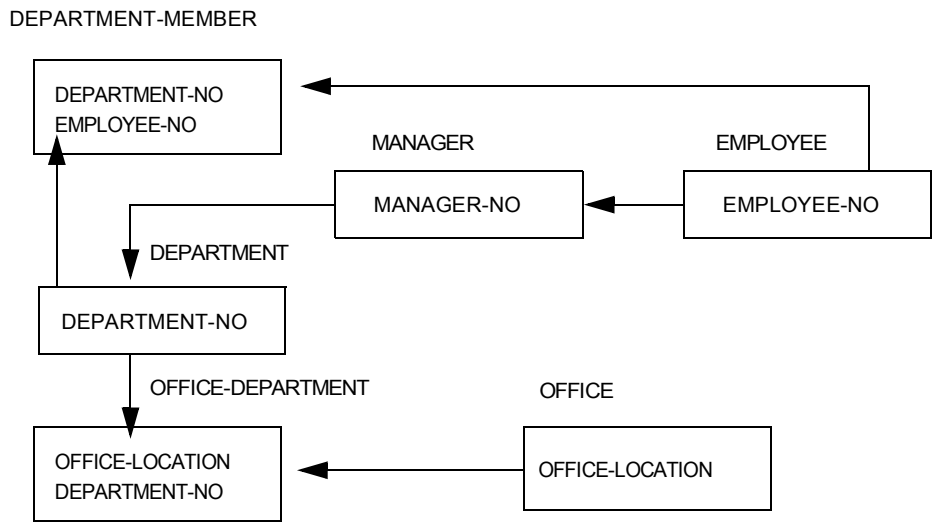
The six tables in the generated DB2 design are shown below. The columns comprising the primary key of the table are shown in capital letters. The non-key columns of the table are shown in lower case.

Table 4 Department Model Example: Contents of the Tables in the Generated DB2 Design

Table Name	Columns Comprising Table		
EMPLOYEE	EMPLOYEE-NO	employee-name	
MANAGER	MANAGER-NO		
DEPARTMENT	DEPARTMENT-NO	department-name	manager-no
DEPARTMENT-MEMBER	DEPARTMENT-NO	EMPLOYEE-NO	
OFFICE	OFFICE-LOCATION	office-name	
OFFICE-DEPARTMENT	OFFICE-LOCATION	DEPARTMENT-NO	

Figure 4 represents the generated DB2 design. It displays all six tables with their respective primary key columns and the foreign key relationships in which they participate.

Figure 4 • Department Model Example: Diagram Showing the Relationships of the Tables in the Generated DB2 Design



The contents of the tables are described below, including each primary key, non-prime columns (if any), and any foreign key relationships in which it participates as either the parent or the dependent table.

EMPLOYEE is a parent table. It has:

- The primary key EMPLOYEE-NO
- The non-prime column employee-name
- The dependent table DEPARTMENT-MEMBER in which the primary key component, EMPLOYEE-NO, is the foreign key to EMPLOYEE
- The dependent table MANAGER in which the primary key, MANAGER-NO, is the foreign key to EMPLOYEE.

MANAGER is both a parent table and a dependent table. It has:

- The primary key, MANAGER-NO, which is the only column in the table
- The dependent table DEPARTMENT in which the non-prime column, manager-no, is the foreign key to MANAGER
- The parent table EMPLOYEE. In this relationship, the primary key MANAGER-NO (or MANAGER) is the foreign key to EMPLOYEE.

DEPARTMENT is both a parent table and a dependent table. It has:

- The primary key, DEPARTMENT-NO
- The non-prime columns department-name and manager-no
- The dependent table DEPARTMENT-MEMBER in which the primary key component, DEPARTMENT-NO, is the foreign key to DEPARTMENT
- The dependent table OFFICE-DEPARTMENT in which the primary key component, DEPARTMENT-NO, is the foreign key to DEPARTMENT
- The parent table MANAGER. In this relationship, the non-prime column manager-no (of DEPARTMENT) is the foreign key to MANAGER.

DEPARTMENT-MEMBER is a dependent table. It has:

- A composite primary key with DEPARTMENT-NO and EMPLOYEE-NO as the component prime columns
- The parent table DEPARTMENT. In this relationship, the prime column DEPARTMENT-NO (of DEPARTMENT-MEMBER) is the foreign key to DEPARTMENT.
- The parent table EMPLOYEE. In this relationship, it is the prime column EMPLOYEE-NO (of DEPARTMENT-MEMBER) that is the foreign key to EMPLOYEE.

OFFICE-DEPARTMENT is a dependent table. It has:

- A composite primary key with OFFICE-LOCATION and DEPARTMENT-NO as the component prime columns
- The parent table DEPARTMENT. In this relationship, the prime column DEPARTMENT-NO (of OFFICE-DEPARTMENT) is the foreign key to DEPARTMENT.
- The parent table OFFICE. In this relationship, it is the prime column OFFICE-LOCATION (of OFFICE-DEPARTMENT) that is the foreign key to OFFICE.

OFFICE is a parent table. It has:

- The primary key OFFICE-LOCATION
- The non-prime column office-name
- The dependent table OFFICE-DEPARTMENT in which the primary key component, OFFICE-LOCATION, is the foreign key to OFFICE.

So far, the example has illustrated how a DB2 design is generated in the WBDA. When you enter the DB2 PREVIEW or DB2 POPULATE command, dictionary definitions are generated automatically for the six DB2 tables, with their respective primary keys and foreign keys.

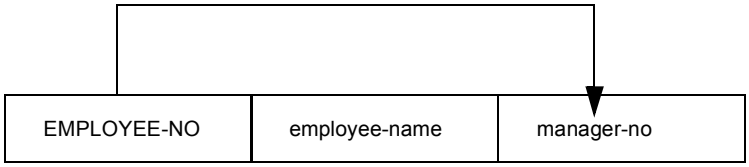
Once the definitions of these tables are in the dictionary you may wish to add more clauses to them. You may also wish to refine some of the generated tables for performance reasons; for instance, you may decide to split or combine some tables.

Let us examine the table MANAGER, which may seem strange because it has only one column. In the example, there is an implied functional dependency from EMPLOYEE-NO (the key of the table EMPLOYEE) to MANAGER-NO (the key of this table). Therefore, it could be useful to merge MANAGER into EMPLOYEE, so that EMPLOYEE would contain an additional column of manager-no.

Then each row in the table containing a value of EMPLOYEE-NO for an employee would contain a different value of manager-no for the employee's manager. Manager-no then would be a foreign key because each value of manager-no would also appear (in a different row) as a value of EMPLOYEE-NO. Thus the table EMPLOYEE would become a self-referencing table; that is, it would be both parent and dependent in the same relationship.

This is shown in Figure 5:

Figure 5 • Department Figure Example: The Table EMPLOYEE as a Self-Referencing Table



The table EMPLOYEE would then become the parent of the table DEPARTMENT, where the column manager-no in DEPARTMENT would be the foreign key to EMPLOYEE.

On the other hand, you may prefer to keep the table MANAGER, because it ensures integrity by containing a list of valid managers. That is, when inserting a new row into the table DEPARTMENT, the referential integrity constraints will validate the manager number by checking it against the MANAGER table. You may also feel that MANAGER, containing a list of valid managers, is useful in its own right; in the future you may have plans to add new attributes or relationships to the entity MANAGER.

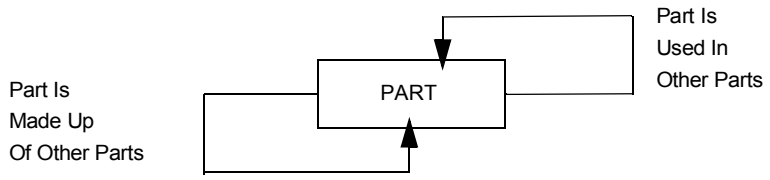
The final decisions about operation and performance are made by the database designer. You can then complete the physical design or the DB2 database by documenting those decisions in the relevant DB2-TABLE, DB2-INDEX, and DB2-VIEW dictionary members.

Parts Model Example

The Bill of Materials (BOM) problem (often referred to as the parts explosion problem) is one of the most common in the manufacturing industry, arising from the need to distinguish between large objects and the smaller objects of which they are composed.

In this example, there is an entity called PART. Its relationship with other manufacturing parts is represented as shown in Figure 6.

Figure 6 • Parts Model Example: The Relationship of PART with other Manufacturing Parts



Begin by defining two ENTITY members in the dictionary, PART-ENT and COMPONENT-ENT. Define the subentities MAJOR-PART-ENT and MINOR-PART-ENT as subentities of PART-ENT.

```
PART-ENT
ENTITY
IDENTIFIER ISPART-NO
ONE-ATTRIBUTES AREPART-NAME, PART-PRICE, PART-QTY-IN-STOCK
SUB-ENTITIES AREMAJOR-PART-ENT, MINOR-PART-ENT

MAJOR-PART-ENT
ENTITY
IDENTIFIER ISMAJOR-PART-NO

MINOR-PART-ENT
ENTITY
IDENTIFIER ISMINOR-PART-NO

COMPONENT-ENT
ENTITY
IDENTIFIER ISMAJOR-PART-NO, MINOR-PART-NO
ONE-ATTRIBUTES AREASSEMBLY-QUANTITY
```

where the entity PART-ENT represents a part in a manufacturing assembly. It has:

- The identifier (or key attribute) PART-NO
- One-attributes (non-prime attributes) PART-NAME, PART-PRICE, and PART-QTY-IN-STOCK
- The two sub-entities MAJOR-PART-ENT and MINOR-PART-ENT.

The two sub-entities are also parts, but each assumes a special role in the assembly of a part; MAJOR-PART-ENT represents a larger part made up of smaller parts, and MINOR-PART-ENT represents a smaller part used in the assembly of larger parts.

The identifier of MAJOR-PART-ENT is MAJOR-PART-NO, and the identifier of MINOR-PART-ENT is MINOR-PART-NO. The domain (the set of all valid values) of each of MAJOR-PART-NO and MINOR-PART-NO is a subdomain or subset of the domain of PART-NO.

In summary, each part is identified by a part number and has as its attributes a part name, a part price, and a part quantity in stock. Also, a part can assume the role of a major part composed of minor parts, or the role of a minor part used in the assembly of a major part.

The entity COMPONENT-ENT represents the composition of the components in an assembly. It states that, for each component (that is, for each major part), there is a fixed number of each of the minor parts used in its assembly. The entity COMPONENT-ENT has:

- The composite identifier consisting of the attributes MAJOR-PART-NO and MINOR-PART-NO
- The one-attribute assembly-quantity, specifying the number of each minor part used in the assembly of the major part. When the entities PART-ENT and COMPONENT-ENT are merged into the WBDA using the MERGE command, these dependencies are derived:

Table 5 Parts Model Example: Derived Dependencies

Dependency			Derived from Entity
PART-NO	→	PART-NAME	PART-ENT
PART-NO	→	PART-PRICE	PART-ENT
PART-NO	→	PART-QTY-IN-STOCK	PART-ENT
MAJOR-PART-NO	⇢	PART-NO	PART-ENT
MINOR-PART-NO	⇢	PART-NO	PART-ENT
MAJOR-PART-NO	→	ASSEMBLY-QUANTITY	COMPONENT-ENT
MINOR-PART-NO			

You can now normalize the dependencies of the composite view, identify potential keys, and generate relations, using the DESIGN command. The generated relations constitute the relational schema.

Refer to the *ASG-DesignManager User's Guide* for details of how dependencies are derived from ENTITY definitions.

The relational schema generated by the DESIGN command for the parts explosion problem contains four relations representing the entities PART-ENT, MAJOR-PART-ENT, MINOR-PART-ENT, and COMPONENT-ENT, as well as the generated dependencies in the WBDA.

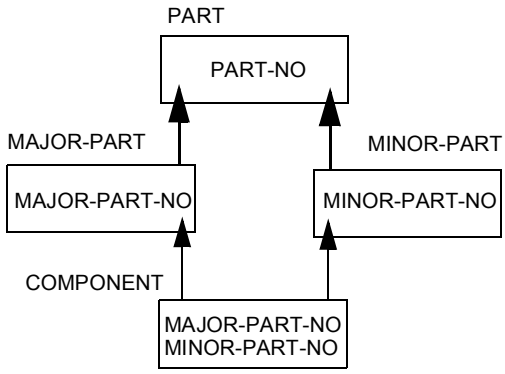
Table 6 shows the data elements in each relation. The names of data elements comprising the key of each relation are shown in capitals. The names of the non-prime data elements are shown in lower case.

Table 6 Parts Model Example: Table of Relations and their Component Data Elements

Relation	Data Elements Comprising Relation			
PART	PART-NO	part-name	part-price	part-qty-in-stock
MAJOR-PART	MAJOR-PART-NO	part-no		
MINOR-PART	MINOR-PART-NO			
COMPONENT	MAJOR-PART-NO	MINOR-PART-NO	assembly-quantity	

Figure 7 shows the four relations with their respective primary key data elements and the foreign key associations, domain associations, and hierarchical-one associations in which they participate.

Figure 7 • Parts Model Example: Diagram Showing Associations of the Relations



The contents of the relations are described below, including, for each, its key, its non-prime data elements (if any) and any associations in which it participates as the source relation.

PART is an FD relation. It has:

- The key PART-NO
- The non-prime data elements part-name, part-price, and part-qty-in-stock
- No associations leading from it.

MAJOR-PART and MINOR-PART are both role relations, with:

- The respective keys MAJOR-PART-NO and MINOR-PART-NO
- The non-prime data element part-no, appearing in each
- The domain association leading from each to the target relation PART, via the domain dependencies from their respective keys, to PART-NO (the key of PART).

COMPONENT is an FD relation. It has:

- A composite primary key consisting of the prime data elements MAJOR-PART-NO and MINOR-PART-NO
- The non-prime data element assembly-quantity
- A hierarchical-one association leading from it to the target relation MAJOR-PART, via its prime data element MAJOR-PART-NO, which is also the key of MAJOR-PART
- A hierarchical-one association leading from it to the target relation MINOR-PART, via its other prime data element MINOR-PART-NO, which is also the key of MINOR-PART.

Refer to the *ASG-DesignManager User's Guide* for details of the structure of the relational schema.

You can now generate the DB2 design from the relational schema, by issuing one of these commands:

- DB2 LIST
- DB2 REPORT
- DB2 PLOT
- DB2 PREVIEW
- DB2 POPULATE.

The DB2 design generated from the relations PART, MAJOR-PART, MINOR-PART, and COMPONENT contains four tables.

When a table in the DB2 design is generated from a relation in the relational schema, the table takes the name and WBDA number of its source relation. In general, all of the data elements of the relation become the columns of the table, where the key of the relation becomes the primary key of the table.

However, in the Parts Model example, two relations prove to be exceptions to this rule. That is, the role relations MAJOR-PART and MINOR-PART both have the non-prime data element part-no, which does not become a column in either of the corresponding tables. This is because the set of valid values of each of MAJOR-PART-NO and MINOR-PART-NO is a subset of the set of valid values of part-no. If part-no were included, each row of the tables MAJOR-PART and MINOR-PART would contain the same value in both the primary key column and the non-prime column.

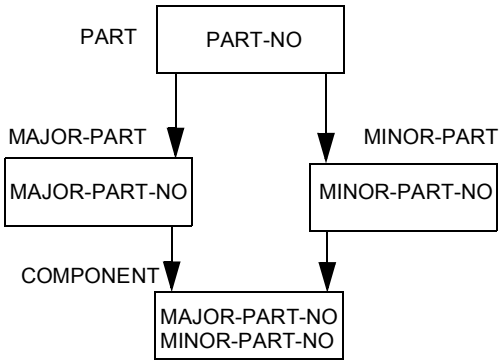
The four tables in the generated DB2 design are shown below. The columns comprising the primary key of the table are shown in capital letters. The non-key columns of the table are shown in lower case.

Table 7 Department Model Example: Contents of the Tables in the Generated DB2 Design

Table Name	Columns Comprising Table			
PART	PART-NO	part-name	part-price	part-qty-in-stock
MAJOR-PART	MAJOR-PART-NO			
MINOR-PART	MINOR-PART-NO			
COMPONENT	MAJOR-PART-NO	MINOR-PART-NO	assembly-quantity	

Figure 8 represents the generated DB2 design. It displays all four tables with their respective primary key columns and the foreign key relationships in which they participate.

Figure 8 • Parts Model Example: Diagram Showing the Relationship of the Tables in the Generated DB2 Design



The following describes the contents of each table, including its primary key, its non-prime columns (if any), and any foreign key relationships in which it participates (as either the parent or the dependent table).

PART is a parent table. It has:

- The primary key PART-NO
- The non-prime columns part-name, part-price, and part-qty-in-stock
- The dependent table MAJOR-PART in which the primary key, MAJOR-PART-NO, is a foreign key to PART
- The dependent table MINOR-PART in which the primary key, MINOR-PART-NO, is also a foreign key to PART.

MAJOR-PART and MINOR-PART are both parent tables and dependent tables, with:

- Respective primary keys MAJOR-PART-NO and MINOR-PART-NO
- The dependent table COMPONENT in which the primary key components, MAJOR-PART-NO and MINOR-PART-NO, are the respective foreign keys to MAJOR-PART and MINOR-PART
- The parent table PART. The primary keys, MAJOR-PART-NO and MINOR-PART-NO (of MAJOR-PART and MINOR-PART) are foreign keys in their respective relationships with the parent table PART.

COMPONENT is a dependent table. It has:

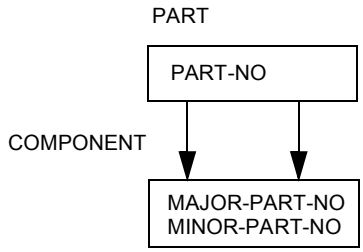
- A composite primary key with MAJOR-PART-NO and MINOR-PART-NO as the constituent prime columns
- The non-prime column assembly-quantity
- The parent table MAJOR-PART. In this relationship, the prime column MAJOR-PART-NO (of COMPONENT) is the foreign key to MAJOR-PART.
- The parent table MINOR-PART. In this relationship, the prime column MINOR-PART-NO (of COMPONENT) is the foreign key to MINOR-PART.

So far the example has illustrated how a DB2 design is generated in the WBDA. When you enter the DB2 PREVIEW or DB2 POPULATE command, DB2 dictionary definitions are generated automatically for the four DB2 tables, with their respective primary keys and foreign keys.

Once the DB2 definitions are in the dictionary, you may wish to add more clauses to them. You may also wish to refine some of the generated tables for performance reasons; for instance, you may decide to split or combine some tables.

The generated tables MAJOR-PART and MINOR-PART both contain only a single column, MAJOR-PART-NO and MINOR-PART-NO, respectively. You may prefer to merge both of these tables with the table COMPONENT, so that the table PART would become the parent of COMPONENT instead of being the parent of MAJOR-PART and MINOR-PART; each of the columns MAJOR-PART-NO and MINOR-PART-NO in the table COMPONENT (the columns comprising its composite primary key) would then become foreign keys of the table PART, as shown in Figure 9:

Figure 9 • Result of Merging the Tables MAJOR-PART and MINOR-PART into the Table COMPONENT



However, it may be useful to keep either or both of MAJOR-PART and MINOR- PART as tables in their own right. This would ensure integrity by maintaining valid lists of both major parts and minor parts.

Thus, when a new row is inserted into the table COMPONENT, the columns for MAJOR-PART and MINOR-PART can be checked for valid values of major parts and minor parts. This can be done either by maintaining the single column tables for MAJOR-PART and MINOR-PART and letting DB2 ensure the integrity, or by programming the checks into your applications.

Furthermore, if you expect subsequently to add new attributes or relationships to either of the entities MAJOR-PART or MINOR-PART, then you should keep the MAJOR-PART and MINOR-PART tables.

The final decisions about operation and performance are made by the database designer. You can then complete the physical design of the DB2 database by documenting those decisions in the relevant DB2-TABLE, DB2-INDEX, and DB2-VIEW dictionary members.

DB2 Command Output

Output from the DB2 REPORT Command

Introduction

The DB2 REPORT command produces the DB2 Table Report that shows the DB2 tables generated in the DB2 design. You can report all the tables in the WBDA or a selection of tables. Tables may be selected by name or WBDA number.

The output includes the total number of tables in the WBDA, details of the contents of each selected table, a description of each dependency represented by the table, and the origin of that dependency. For each foreign key relationship in which the table participates, information about the related parent or dependent table is given.

If you have the optional User Formatted Output facility installed, you can tailor the format and content of the DB2 Table Report.

Contents of Tables

For each selected table, the DB2 Table Report shows:

- The table type, that is, PARENT/ROOT, DEPENDENT/PARENT, DEPENDENT/LEAF, or INDEPENDENT
- The table number
- The table name (if one has been assigned to the corresponding relation in the relational schema)
- The columns comprising the primary key of the table
- The non-prime columns present in the table (if any); that is, each column in the table which does not form any part of the key.

Two report lines are output for each dependency represented by the table. The first output line shows:

- Its absolute dependency number in the WBDA (ABS REL column)
- The data elements comprising its left-hand side
- The data elements comprising its right-hand side
- The dependency type: functional (FD), multivalued (MVD), or domain (DD).

The second output line shows the origin of the dependency:

- Either generated by the MERGE command from one or more data-views; if so, the report includes the name and type (USERVIEW or ENTITY) of each data-view, and the relative number of the dependency in that data-view
- Or it was created during MERGE command processing as an implied FD.

Refer to the *ASG-DesignManager User's Guide* for details of dependencies and of generating implied FDs from GROUP data elements.

Foreign Key Relationships

For each selected table in the DB2 design which participates in foreign key relationships, the DB2 Table Report gives additional information about each of its related parent or dependent tables.

In the output for a selected table, the two possible types of foreign key relationship are indicated by the following keywords:

- FOREIGN-KEY is used to indicate a *direct* foreign key relationship, derived either from a direct foreign key association or from a direct hierarchical-one association in the corresponding relational schema
- DOMAIN indicates a *domain* type of relationship, derived from a domain association.

The significance of the relationship types is explained below, both for a related parent table and for a related dependent table. (In each case, T1 is the selected table, T2 is its related parent table, and T3 is its related dependent table.)

For each related parent table:

- FOREIGN-KEY indicates a direct foreign key relationship between T1 and T2, where the primary key of T2 is contained as a set of columns in T1 (that is, as a foreign key in T1 to the parent table T2), and there is no intermediate table that is both a parent of T1 and a dependent of T2 in direct foreign key relationships
- DOMAIN indicates a domain relationship between T1 and T2, due to a domain dependency in the WBDA holding from the primary key of T1 to the primary key of T2.

For each related dependent table:

- FOREIGN-KEY indicates a direct foreign key relationship between T1 and T3, where the primary key of T1 is contained as a set of columns in T3 (that is, as a foreign key to T1 in the dependent table T3), and there is no intermediate table that is both a parent of T3 and a dependent of T1 in direct foreign key relationships
- DOMAIN indicates a domain relationship between T1 and T3, due to a domain dependency in the WBDA holding from the primary key of T3 to the primary key of T1.

The DB2 Table Report contains the following information for each parent table of a table selected for reporting:

- The parent table number
- The parent table name (if one has been assigned)
- The type of relationship (a direct or domain type of foreign key relationship)
- The columns comprising the primary key of the parent table
- The columns of the selected table comprising the foreign key to the parent table, where:
  - In a direct foreign key relationship, these columns are identical to those comprising the primary key of the parent table
  - In a domain relationship, they are different.

Each foreign key column in the selected table is shown paired with its corresponding primary key column in the parent table.

The DB2 Table Report contains the following information for each dependent table of a table selected for reporting:

- The dependent table number
- The dependent table name (if one has been assigned)
- The type of relationship, that is, a direct or domain type of foreign key relationship
- The columns comprising the primary key of the dependent table
- The columns of the dependent table comprising the foreign key to the selected table, where:
  - In a direct foreign key relationship, these columns are identical to the columns comprising the primary key of the selected table
  - in a domain relationship, they are different.

Each foreign key column in the dependent table is shown paired with its corresponding primary key column in the selected table.

Example

In this example, a DB2 Table Report is produced for the following tables, which have been generated in the DB2 design in the WBDA. The names of the columns comprising the primary key of each table are shown in capital letters. The names of non-prime columns are shown in lower case.

Table 8 Tables in the DB2 Design in the WBDA

Table No. and Name		Columns Comprising Table		
1	DEPARTMENT-MEMBER	DEPARTMENT-NO	EMPLOYEE-NO	
2	OFFICE-DEPARTMENT	OFFICE-LOCATION	DEPARTMENT-NO	
3	DEPARTMENT	DEPARTMENT-NO	department-name	manager-no
4	EMPLOYEE	EMPLOYEE-NO	employee-name	
5	OFFICE	OFFICE-LOCATION	office-name	
6	MANAGER	MANAGER-NO		

The tables in the DB2 Table Report (Figure 10) are described in order of table number, as they appear in the list above.

Figure 10 • Example of DB2 REPORT Output

```
**************************************************
*       *
*       DB2 TABLE REPORT   *
*       *
*       TOTAL NUMBER OF TABLES  ...6*
*       *
**************************************************

==================LEAF/DEPENDENT==================
1 DEPARTMENT-MEMBER
==================================================
PRIMARY KEYDATA ELEMENTS
DEPARTMENT-NO
EMPLOYEE-NO
```

ABS REL DATA-VIEW	LEFT-HAND-SIDE	TYPE	RIGHT-HAND-SIDE
5 3 ENTITY	DEPARTMENT-NO	—MVD->	>EMPLOYEE-NO DEPARTMENT-ENT

Parent Table		Current Table Foreign Key
NAME	3 DEPARTMENT	
TYPE	FOREIGN-KEY	
P-KEY	DEPARTMENT-NO	DEPARTMENT-NO
NAME	4 EMPLOYEE	
TYPE	FOREIGN-KEY	
P-KEY	EMPLOYEE-NO	EMPLOYEE-NO

```
==================LEAF/DEPENDENT==================
2  OFFICE-DEPARTMENT
==================================================
PRIMARY KEYDATA ELEMENTS
OFFICE-LOCATION
DEPARTMENT-NO
```

ABS	LEFT-HAND-SIDE	TYPE RIGHT-HAND-SIDE
REL DATA-VIEW		
7	OFFICE-LOCATION—MVD-	››DEPARTMENT-NO
2 ENTITY	OFFICE-ENT	

Parent Table		Current Table Foreign Key
NAME	3 DEPARTMENT	
TYPE	FOREIGN-KEY	
P-KEY	DEPARTMENT-NO	DEPARTMENT-NO
NAME	5 OFFICE	
TYPE	FOREIGN-KEY	
P-KEY	OFFICE-LOCATION	OFFICE-LOCATION

```
==================PARENT/DEPENDENT==================
3  DEPARTMENT
====================================================
PRIMARY KEYDATA ELEMENTS
DEPARTMENT-NO
        DEPARTMENT-NAME
        MANAGER-NO
```

ABS REL DATA-VIEW	LEFT-HAND-SIDE	TYPE	RIGHT-HAND-SIDE
3	DEPARTMENT-NO—FD-->DEPARTMENT-NAME		
1 ENTITY	DEPARTMENT-ENT		
4	DEPARTMENT-NO—FD-->MANAGER-NO		
2 ENTITY	DEPARTMENT-ENT		

Parent Table		Current Table Foreign Key
NAME	6 MANAGER	
TYPE	FOREIGN-KEY	
P-KEY	MANAGER-NO	MANAGER-NO

Dependent Table		Current Table Foreign Key
NAME	1 DEPARTMENT-MEMBER	
TYPE	FOREIGN-KEY	
P-KEY	DEPARTMENT-NO	
	EMPLOYEE-NO	
F-KEY	DEPARTMENT-NO	DEPARTMENT-NO
NAME	2 OFFICE-DEPARTMENT	
TYPE	FOREIGN-KEY	
P-KEY	OFFICE-LOCATION	
	DEPARTMENT-NO	
F-KEY	DEPARTMENT-NO	DEPARTMENT-NO

```
==================ROOT/PARENT==================
4  EMPLOYEE
===============================================
PRIMARY KEYDATA ELEMENTS
EMPLOYEE-NO
        EMPLOYEE-NAME
```

ABS REL DATA-VIEW	LEFT-HAND-SIDE TYPE RIGHT-HAND-SIDE
1 1 ENTITY	EMPLOYEE-NO—FD-->EMPLOYEE-NAME EMPLOYEE-ENT

Parent Table		Current Table Primary Key
NAME	1 DEPARTMENT-MEMBER	
TYPE	FOREIGN-KEY	
P-KEY	DEPARTMENT-NO	
	EMPLOYEE-NO	
F-KEY	EMPLOYEE-NO	EMPLOYEE-NO
NAME	6 MANAGER	
TYPE	DOMAIN	
P-KEY	MANAGER-NO	
F-KEY	MANAGER-NO	EMPLOYEE-NO

```
==================ROOT/PARENT==================
5  OFFICE
===============================================
PRIMARY KEYDATA ELEMENTS
OFFICE-LOCATION
        OFFICE-NAME
```

ABS REL DATA-VIEW	LEFT-HAND-SIDE TYPE RIGHT-HAND-SIDE
6 1 ENTITY	OFFICE-LOCATION—FD-->OFFICE-NAME OFFICE-ENT

Dependent Table		Current Table Foreign Key
NAME	2 OFFICE-DEPARTMENT	
TYPE	FOREIGN-KEY	
P-KEY	OFFICE-LOCATION	
	DEPARTMENT-NO	
F-KEY	OFFICE-LOCATION	OFFICE-LOCATION

```
==================PARENT/DEPENDENT==================
6  MANAGER
====================================================
PRIMARY KEYDATA ELEMENTS
MANAGER-NO
```

ABS REL DATA-VIEW	LEFT-HAND-SIDE TYPE RIGHT-HAND-SIDE
2 2 ENTITY	MANAGER-NO===DD==>EMPLOYEE-NO EMPLOYEE-ENT

Parent Table		Current Table Foreign Key
NAME	4 EMPLOYEE	
TYPE	DOMAIN	
P-KEY	EMPLOYEE-NO	MANAGER-NO

Dependent Table		Current Table Primary Key
NAME	3 EMPLOYEE	
TYPE	FOREIGN-KEY	
P-KEY	DEPARTMENT-NO	
F-KEY	MANAGER-NO	MANAGER-NO

```
**************************************************
*            *
*                  END OF SQL REPORT*
*           *
**************************************************
```

Output from the DB2 PLOT CLUSTER Command

Introduction

The output of the DB2 PLOT CLUSTER command, called the DB2 Cluster plot, provides you with detailed graphical displays for selected tables in the DB2 design in the WBDA.

The output shows where relationships exist between tables, the type of the relationships, and the primary and foreign keys used.

For each selected table, a diagram is produced in cluster form, showing the table's foreign key relationships (if any) with the other tables of the DB2 design. You can output cluster diagrams for all the tables in the WBDA or for tables selected by name or WBDA number.

After all the cluster diagrams have been displayed, the DB2 Design Relationship matrix is output; this is a two-dimensional table which summarizes all of the foreign key relationships between the tables in the WBDA. The DB2 Design Relationship matrix always shows all the tables in the WBDA.

If you have the optional User Formatted Output facility installed, you can specify the name of an appropriate FORMAT member of the dictionary in the command in order to tailor the format and content of the DB2 Cluster plot.

Format of the Cluster Diagram

In each cluster diagram displayed in the DB2 Cluster plot, the table selected for output is depicted by a larger box and the related tables by smaller boxes. Foreign key relationships between tables are represented by connecting arrows. The DB2 convention is followed, in which the arrow points from the parent table to the dependent table.

The type of foreign key relationship existing between the selected table (T1) and each related table (T2) is indicated by the type of arrow used to connect them, as explained below.

Direct foreign key relationships are indicated either by:

T1 ⟶ T2

where the selected table is the parent table in the relationship; that is, the primary key of T1 is contained as a set of columns in T2, or by:

T1 ⟵ T2

where the selected table is the dependent table in the relationship; that is, the primary key of T2 is contained as a set of columns in T1.

If the selected table is related both as a parent and as a dependent table in two different relationships with the same table, then the bidirectional arrow is used to represent the relationship:

T1 ⟷ T2

A domain relationship is indicated either by:

```
T1 =====> T2
```

where the selected table is the parent table in the relationship; that is, a domain relationship exists between T1 and T2, due to a domain dependency in the WBDA holding from the primary key of T2 to the primary key of T1, or by:

```
T1 <===== T2
```

where the selected table is the dependent table in the relationship; that is, a domain relationship exists between T1 and T2, due to a domain dependency in the WBDA holding from the primary key of T1 to the primary key of T2.

In a cluster the tables are arranged vertically. In accordance with ASG-DesignManager convention, related parent tables appear above the selected table and related dependent tables appear below it. The only exception to this convention occurs when a foreign key relationship holds in both directions between the selected table and a related table, in which case the related table appears below the selected table.

The Information in the Cluster Diagram

Each cluster diagram of the DB2 Cluster plot displays information about the selected table and about each of its related tables.

The following information is given for the selected table:

- The table type; that is, ROOT/PARENT, PARENT/DEPENDENT, LEAF/DEPENDENT, or INDEPENDENT
- The table's WBDA number
- The table name (the same name as that of its corresponding WBDA relation, if that relation has previously been named via the NAME command)
- The names of the columns comprising the primary key of the table
- The name of each non-prime column in the table, that is, each column which does not form part of the key.

The following information is given for each related table:

- The table WBDA number
- The table name (if its corresponding relation has been named using the NAME command)
- The names of the columns comprising the primary key of the table
- The relationship that holds between the selected table and the related table (represented by a connecting arrow).

Refer to "Format of the Cluster Diagram" on page 64 for information about directional arrows. Refer to "Introduction to Examples" on page 37 for details of the case study on which this example is based.

Example of the Output

In Table 9, the names of columns which form the primary key of each table are shown in capital letters, and the names of non-prime columns are shown in lower case.

Refer to "Introduction to Examples" on page 37 for details of the case study on which this example is based.

Table 9 Example Output: Tables and Columns

Table No. and Name		Columns Comprising Table		
1	DEPARTMENT-MEMBER	DEPARTMENT-NO	EMPLOYEE-NO	
2	OFFICE-DEPARTMENT	OFFICE-LOCATION	DEPARTMENT-NO	
3	DEPARTMENT	DEPARTMENT-NO	department-name	manager-no

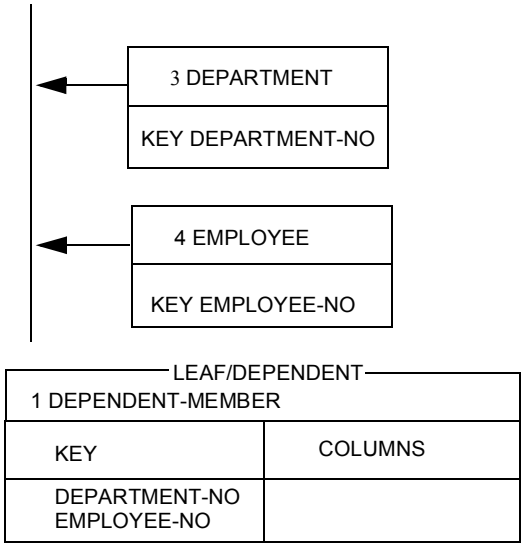
Table 9 Example Output: Tables and Columns

Table No. and Name		Columns Comprising Table	
4	EMPLOYEE	EMPLOYEE-NO	employee-name
5	OFFICE	OFFICE-LOCATION	office-name
6	MANAGER	MANAGER-NO	

Taking each of these tables in turn, different clusters would be displayed in the DB2 Cluster plot.

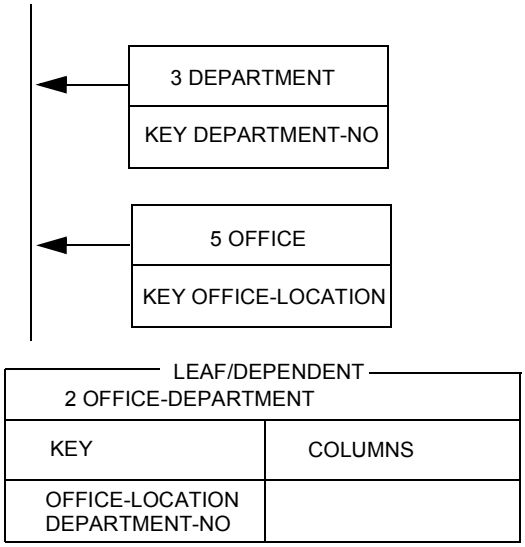
The cluster in Figure 11 displays when the table DEPARTMENT-MEMBER is selected:

Figure 11 • Displayed Cluster in the DEPARTMENT-MEMBER Table



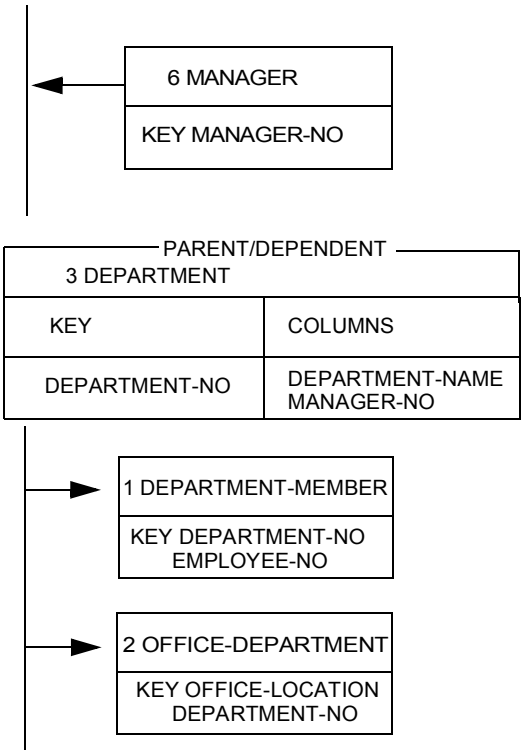
The cluster in Figure 12 displays when the table OFFICE-DEPARTMENT is selected:

Figure 12 • Displayed Cluster in the OFFICE-DEPARTMENT table



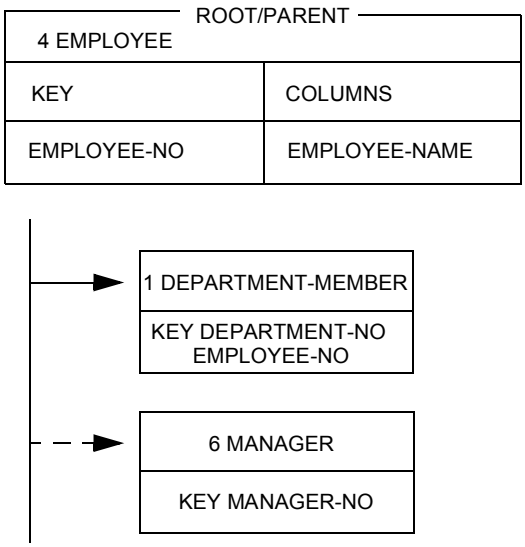
The cluster in Figure 13 displays when the table DEPARTMENT is selected:

Figure 13 • Displayed Cluster in the DEPARTMENT Table



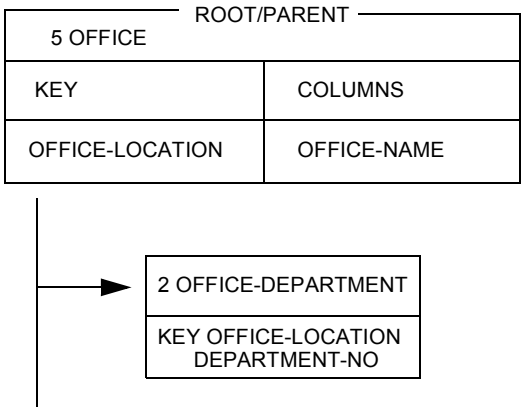
The cluster in Figure 14 displays when the table EMPLOYEE is selected:

Figure 14 • Displayed Cluster in the EMPLOYEE Table



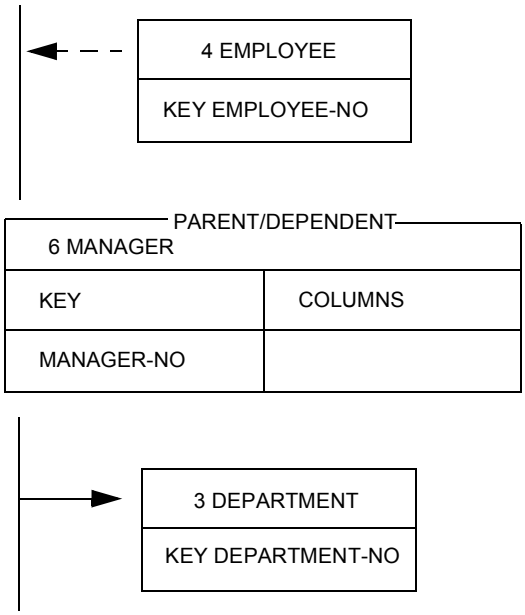
The cluster in Figure 15 displays when the table OFFICE is selected:

Figure 15 • Displayed Cluster in the OFFICE Table



The cluster in Figure 16 displays when the table MANAGER is selected:

Figure 16 • Displayed Cluster in the MANAGER Table



The DB2 Design Relationship Matrix

In the DB2 Cluster plot, the DB2 Design Relationship matrix is output after all the clusters for the selected tables have been displayed.

Note:

The matrix always shows all the tables in the WBDA, regardless of any selections specified in the DB2 PLOT CLUSTER command.

The matrix is a table of entries summarizing all the foreign key relationships between the tables in the WBDA. It can be used as a quick reference to determine the existence of a foreign key relationship and its type.

The matrix is an *n*-by-*n* square array, where n is the total number of tables in the WBDA. Row 1 and column 1 correspond to table 1, row 2 and column 2 to table 2, and so on. The rows of the matrix represent the parent tables, and the columns represent the dependent tables.

The matrix shown in Figure 17 is the one produced after the cluster diagrams discussed earlier.

Figure 17 • Matrix Produced by the DB2 PLOT Command

		1	2	3	4	5	6
DEPARTMENT-MEMBER	1						
		:	:	:	:	:	
OFFICE-DEPARTMENT	2						
		:	:	:	:	:	
DEPARTMENT	3	1	1				
		:	:	:	:	:	
EMPLOYEE	4	1					D
		:	:	:	:	:	
OFFICE	5		1				
		:	:	:	:	:	
MANAGER	6			1			

By reading across a row, you can identify the relationships that hold from a particular parent table to its dependent tables.

By reading down a column, you can identify the relationships that hold to a dependent table from each of its parent tables. A relationship between two tables is indicated by a character at the intersection of the row and column representing those tables. The character used indicates the type of relationship that exists:

- 1 indicates a direct foreign key type of relationship from the row table (the parent table) to the column table (the dependent table)
- D indicates a domain type or foreign key relationship from the row table (the parent table) to the column table (the dependent table). This means that a domain dependency exists in the WBDA holding from the key of the column table to the key of the row table.

For example, in the matrix above you can see that DEPARTMENT is the parent table of DEPARTMENT-MEMBER and OFFICE-DEPARTMENT, and that the relationship is the direct foreign key type.

If no relationship exists between one table and another, then the corresponding matrix intersection is left blank.

Refer to "Introduction to Examples" on page 37 for details of the case study on which this example is based.

Output from the DB2 PLOT REFERENTIAL-STRUCTURES Command

Introduction

The output of the DB2 PLOT REFERENTIAL-STRUCTURES command, called the DB2 Referential Structures plot, provides you with a diagrammatic overview of the referential structures comprising the DB2 design in the WBDA. They are displayed without details of the DB2 table content.

The DB2 Referential Structures plot is complementary to the DB2 Cluster plot, which, for each DB2 table in the DB2 design, provides details of its content and its relationships with other DB2 tables. The DB2 Referential Structures plot gives you an easy-to-understand overall picture of one or more referential structures and the tables contained in each.

The relationships displayed between tables in the DB2 Referential Structures plot include direct foreign key relationships and domain relationships.

The tables displayed in the DB2 Referential Structures plot can be of any of these types:

- PARENT/ROOT
- DEPENDENT/PARENT
- DEPENDENT/LEAF
- INDEPENDENT.

In a DB2 Referential Structures plot, the tables are displayed as boxes and the relationships as connecting lines, where these lines appear as unidirectional arrows. However, in a plot for a large and complex DB2 design, you would find that the number of arrows that cross one another would, in general, create a mass of confusing detail.

In the DB2 Referential Structures plot, this is avoided completely by the tactic of displaying the boxes and lines in the form of an equivalent hierarchical (tree) structure. The composition of the (relational) DB2 design is not affected, only the way it is represented. The great advantage of the hierarchical display is that connecting arrows will never cross, no matter how large and complex the DB2 design. This makes it much easier for you to perceive its overall structure.

The tree structure starts with a single box, the *seed*, and branches out hierarchically to the right in columnar levels. Successive levels consist of vertically aligned *child* boxes, each connected to a box at the preceding level. The first level consists of only the seed. The second level contains children of the seed, the third level contains children of the level two boxes, and so on. A box that has no child at the next level is called a *leaf*. The children of a non-leaf box plus its childrens' children, and so on, are called its *descendants*.

In addition to the representation of the DB2 design as an equivalent hierarchical structure, a further simplifying feature has been incorporated in the DB2 Referential Structures plot which serves to reduce the number of boxes appearing in the diagram. In the tree structure, no table is represented more than once by a box. Every other occurrence of the table and any sub-branch of lower level tables emanating from it in the tree (that is, its descendants) is represented in the plot by a single pointer instead of duplicating the entire sub-branch for the occurrence.

In a DB2 Referential Structures plot, boxes are displayed with dashed outlines and pointers with dotted outlines. There are a number of command options available to you for the DB2 Referential Structures plot which can further simplify the display. You can show either:

- All the tables of the DB2 design, or
- An individual referential structure based on a specified seed.

You can also specify the direction of (and thus limit) the relationships to be displayed, that is parent or dependent relationships. These options enable you to focus attention on desired subsets of the design and on desired types of access through the design.

If you have the optional User Formatted Output facility installed, you can also specify a meaningful title for the plot by entering it as a string in the ASG-ControlManager command, SET FORMAT-TITLE, before you issue the DB2 PLOT command.

Layout

In summary, in the DB2 Referential Structures plot, the tables in the DB2 design are represented by boxes and pointers; relationships between the tables are represented by connecting lines. In die output medium, boxes are displayed with dashed outlines, pointers with dotted outlines, and connecting lines appear as unidirectional arrows.

Table numbers are displayed on the left lower boundary of the corresponding box or pointer. If a table is named, the interior of a box or pointer contains the table name; otherwise, it contains text formed from the table's primary key.

Text formed from a key consists of up to three lines. Each line of text is formed from a data element contained in the key. If the key contains more than one data element, the lines are formed in alphanumeric order of data element name. Each line is limited to a maximum of eleven characters, comprising either the data element name, or the first 11 characters of the name.

As a consequence, it is possible for the same text to be formed from the keys of different tables. However, the text displayed is intended only as an aid to identification. Positive identification of the table being represented is given by the table number which appears on the left lower boundary of each box or pointer displayed.

Table 10 shows examples of text formed from table keys:

Table 10 Examples of Text Formatted from Table Keys

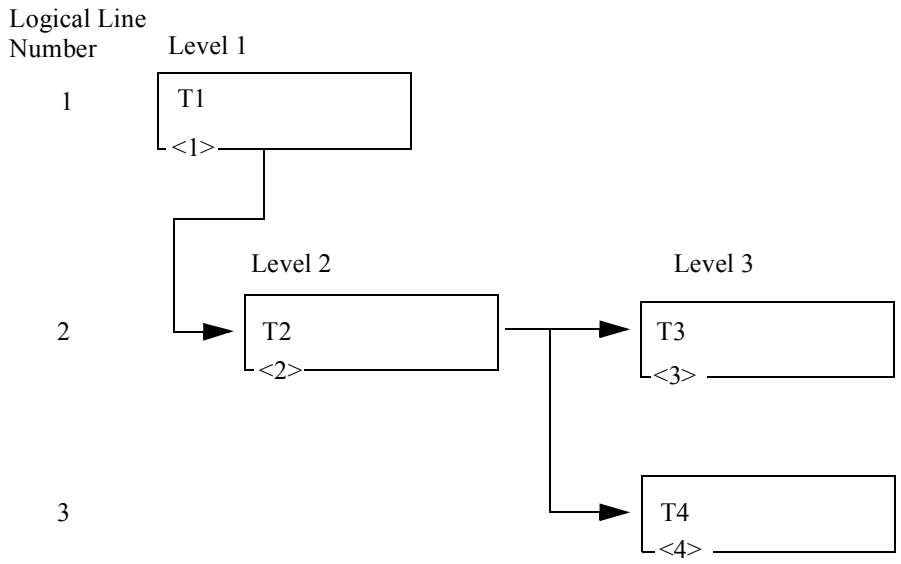
Key	Text Formed	
EMPLOYEE-NO	EMPLOYEE-NO	(no truncation)
EMPLOYEE-NAME	EMPLOYEE-NA	(first 11 characters)
DEPARTMENT-NO,	DEPARTMENT-	(first 11 characters)
EMPLOYEE-NAME,	EMPLOYEE-NA	(first 11 characters)
OFFICE-NO	OFFICE-NO	(no truncation)
DEPARTMENT-NO	DEPARTMENT-	(first 11 characters)
EMPLOYEE-NAME,	EMPLOYEE-NA	(first 11 characters)
OFFICE-NO	OFFICE-NO	(no truncation)
PROJECT-NO		(omitted)

Boxes and their connecting lines are laid out on the output medium in *logical lines*. Each logical line occupies six physical print lines and contains one or more boxes and at most one pointer (perhaps none). Logical lines are numbered consecutively, beginning with one. These numbers are very useful as they are used in pointers and in the Numeric and Alphabetic directories that follow the plot to help you locate any table displayed in the plot.

The boxes and pointers are laid out from left to right on the logical lines in order of the hierarchical levels they form. The highest level is that of the *seed*, which is placed in the upper left-hand corner of the plot. Each lower level box (or pointer) appears to the right of the box (at the next higher level) to which it is connected as a child. All the children of a given box are shown at the next level, one below the other, each connected to the given box.

The seed, which is the only level 1 box in the hierarchy, appears as the only box on logical line number 1. The level 2 boxes consist of the children of the seed, that is all the tables which have direct relationships with the seed. The first of these is placed on logical line 2 at level 2. The level 3 boxes and pointers are the children of the level 2 boxes. If the first level 2 box has any children, the first of these is placed on logical line 2 to the right of (and connected with) the level 2 box. The placement of lower level boxes and pointers on the logical lines follows the same pattern, with the children of a box appearing to the right of the box. A box that has no children is called a *leaf*. A simple example of a DB2 Referential Structures plot appears in Figure 18:

Figure 18 • DB2 Referential Structures Plot



In Figure 18, foreign key relationships are depicted from table T1 (the seed for the plot) to table T2 and from table T2 to each of tables T3 and T4.

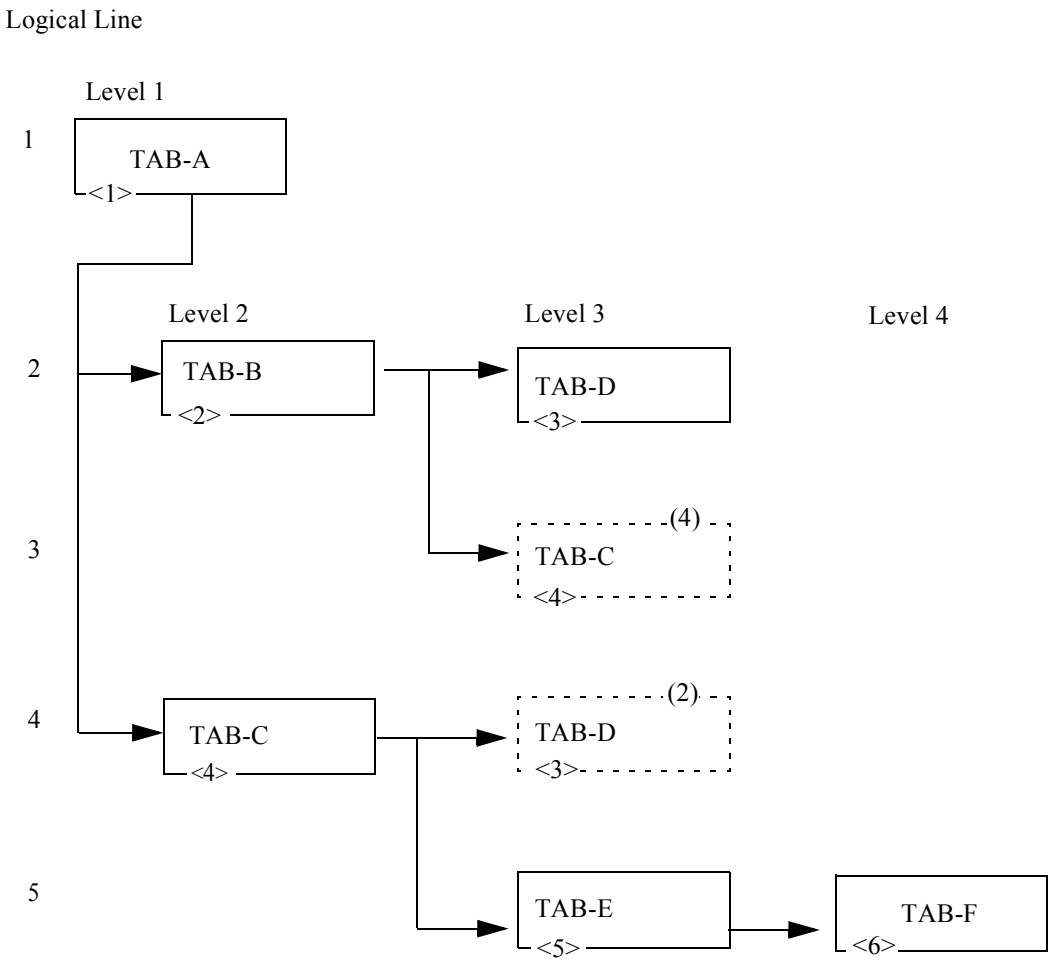
Use of Pointers

In the DB2 Referential Structures plot, a table is represented by a pointer instead of a box if it has already been displayed as a box elsewhere in the plot, either at a higher level on any logical line or at the same level but on a preceding logical line. The purpose is to display each table only once as a box along with any lower level descendants the box may have. Thereafter, the table is displayed as a pointer (without lower level descendants) to the logical line in which it appeared as a box. The number of this logical line always appears on the right upper boundary of the pointer. A table may be displayed as a pointer several times in the plot, but it cannot be displayed more than once as a box.

In fact, a pointer being displayed in a DB2 Referential Structures plot can be quite significant. The appearance of such a pointer serves to highlight one of two important situations in a DB2 design. A pointer signifies that either the table is in a cycle or that the table participates in more than one foreign key relationship; a quick glance at the output will indicate which.

Consider the sample plot appearing in Figure 19:

Figure 19 • Pointers Used in DB2 Referential Structures Plot



In Figure 19, note that:

- Table TAB-C appears as a pointer in level 3 because it was already displayed as a box in level 2. This pointer highlights the fact that TAB-C is a dependent table of both TAB-A and TAB-B.
- Table TAB-D appeal's as a pointer in level 3 on logical line 4 because it appeared as a box in the same level on logical line 2. This pointer highlights the fact that TA B-D is a dependent table of both TA B-B and TAB-C.
- No pointers are required for tables TAB-E and TAB-F because they are descendants of table TA B-C and their repetition is indicated by the pointer for table TAB-C.

There is one other circumstance requiring the use of a pointer. This occurs when there is not enough room on a logical line for all the tables that should be displayed on it. That is, the last box for which there is room has one or more children at the next level (it is not a leaf). In this case, the box is replaced by a pointer and the table it represents is placed in a *continuation seed* list.

Each continuation seed is then processed along with its descendants (that is, its children, its children's children, and so on), if any, further down in the diagram just as if it were the seed for a new Referential Structure plot. That is, it will appear as a box in the seed position at the top of a new page. Pictorially, it will appear as the seed of anew tree. Structurally, however, it will be a continuation of the incomplete branch. In this case, the pointer is called a *continuation pointer*.

A plot which begins with a continuation seed is called a *continuation plot*. It is a continuation of the main plot, which begins with the *primary seed*. Continuation plots appear after the main plot (or after any additional plots). Logical line numbers are assigned consecutively throughout each plot and continue consecutively from one plot to another.

In contrast with pointers, there is only one circumstance in which a box will have an entry on its right upper boundary, that is, when the box is a continuation seed. In this case, the entry is the number of the logical line from which it is continued.

Additional Plots

After the main plot and any continuation plots have been displayed in the DB2 Referential Structures plot (and if the ALL keyword has been specified in the DB2 PLOT command), ASG-DesignManager looks for any additional seeds that may be required to ensure that every table in the DB2 design is displayed. A plot which begins with an additional seed is called an additional plot. (The main plot begins with the primary seed and each continuation plot with a continuation seed.)

An additional plot is not a continuation of the main plot. It represents a separate hierarchy. Separate hierarchies emanating from different seed tables nevertheless can belong to the same referential structure provided that all are linked via tables shared in common. Each such link would be represented by a pointer from one hierarchical plot to a logical line in another hierarchical plot.

Separate hierarchical plots belonging to the same referential structure can appear in the display only when the keywords ALL and either PARENTS or DEPENDENTS are specified in the command. If, on the other hand, ALL is specified but neither PARENTS nor DEPENDENTS (the default selection, indicating that both parent and dependent foreign key relationships are to be plotted), then every additional plot will represent not only a separate hierarchy but also a complete referential structure.

Indeed, not specifying PARENTS or DEPENDENTS in the command (no matter which of the ALL or SEED options has also been selected) is the only way to ensure the display of hierarchies that represent complete referential structures.

The seed for an additional plot appears in the seed position at the top of a new page. The logical line number of the seed follows consecutively from the last logical line of the preceding plot.

You can always distinguish between a continuation seed and an additional seed because the former has a logical line number entered on its right upper boundary whereas the latter does not.

Note:

An additional plot may itself be followed by one or more continuation plots.

It is possible in this process to produce an additional plot which takes the form of a seed-only hierarchy. An additional seed may, for instance, be a leaf with no children. In this case, the additional plot consists of only a single box.

In particular, after all the tables in all the referential structures have been processed, ASG-DesignManager will produce a seed-only hierarchy for each independent table, if any, in the DB2 design.

Use of Directories

At the end of the DB2 Referential Structure plot, two directories are given for referencing the tables displayed, the Numeric directory and the Alphabetic directory. The Numeric directory is ordered by table number and shows, for each table displayed:

- In the first column, the table number and its name (if a name has been assigned)
- In the second column, the logical line number on which it is displayed as a box
- In the third column, every logical line number on which it is displayed as a pointer, indicating each additional instance in which (he table participates in a foreign key relationship.

The Alphabetic directory contains exactly the same information, but only for tables which have been named. They are listed in alphanumeric order of table name.

Figure 20 contains the same sample DB2 Referential Structure plot shown earlier in the discussion of the use of pointers. In addition, the corresponding Numeric and Alphabetic directories are also given.

Figure 20 • Pointers Used in DB2 Referential Structures Plot

Logical Line

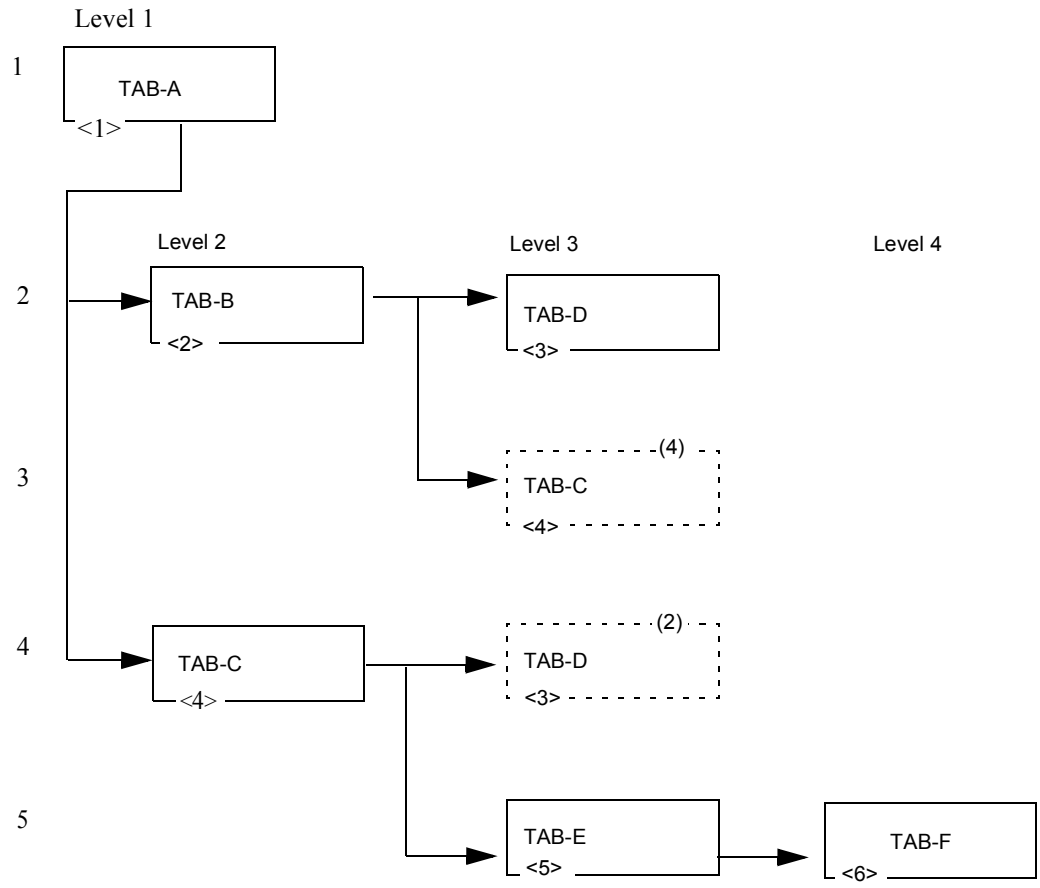


Table 11 is the numeric directory for Figure 20.

Table 11 Numeric Directory

	TABLE	LINE	OTHER OCCURRENCES
1	TAB-A	1	
2	TAB-B	2	
3	TAB-C	2	4
4	TAB-D	4	3
5	TAB-E	5	
6	TAB-F	5	

Table 12 is the alphabetic directory for Figure 20 on page 78

Table 12 Alphabetic Directory

	TABLE	LINE	OTHER OCCURRENCES
1	TAB-A	1	
2	TAB-B	2	
3	TAB-C	4	3
4	TAB-D	2	4
5	TAB-E	5	
6	TAB-F	5	

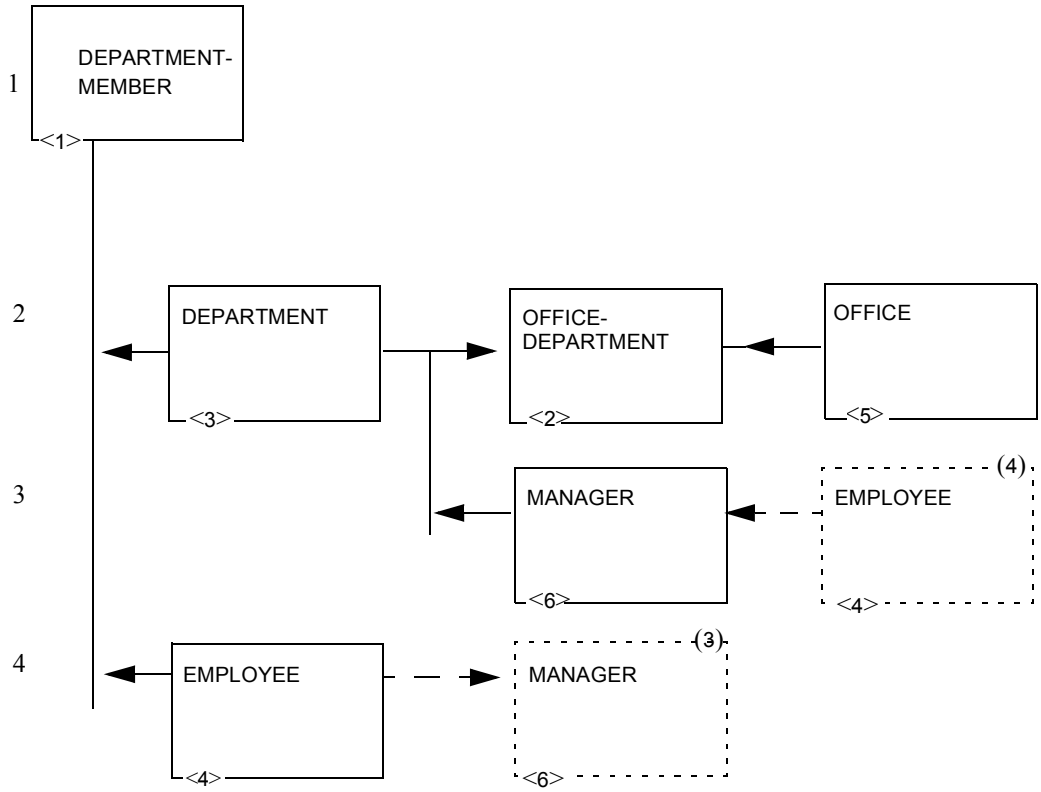
At a glance, you can see in the directories that tables TAB-C and TAB-D have entries in the third column, indicating that they both appear in more than one foreign key relationship. The directories provide another way to distinguish between a continuation seed and an additional seed. A continuation seed will have a single logical line number entered in the third column of the directory indicating the line from which the plot has been continued, whereas an additional seed will have no entry in the third column.

Example

Figure 21 is an example of the output from the DB2 PLOT REFERENTIAL-STRUCTURES command for the Department model, where table number 1, that is table DEPARTMENT-MEMBER, is specified as the seed with the default taken that all relationships are to be displayed.

Figure 21 • DB2 PLOT REFERENTIAL-STRUCTURES Output for Department Model Example

```
*****************************************************************
* D B 2  R E F E R E N T I A L - S T R U C T U R E S  P L O T*
*****************************************************************
```




```
*****************************************************************
*E N D  O F  D B 2  R E F E R E N T I A L - S T R U C T U R E S*
*****************************************************************
```

Table 13 is the numeric directory for Figure 21.

Table 13 Numeric Directory

	TABLE	LINE	OTHER OCCURRENCES
1	DEPARTMENT-MEMBER	1	
2	OFFICE-DEPARTMENT	2	
3	DEPARTMENT	2	
4	EMPLOYEE	4	3
5	OFFICE	2	
6	MANAGER	3	4

Table 14 is the alphabetic directory for Figure 21 on page 80.

Table 14 Alphabetic Directory

	TABLE	LINE	OTHER OCCURRENCES
3	DEPARTMENT	2	
1	DEPARTMENT-MEMBER	1	
4	EMPLOYEE	4	3
6	MANAGER	3	4
5	OFFICE	2	
2	OFFICE-DEPARTMENT	2	

Output from the DB2 LIST TABLES Command

Introduction

The DB2 LIST TABLES command produces a list of all or some of the DB2 tables in the WBDA. For each DB2 table selected, the list includes the WBDA number of the table, its primary key, its name (if one has been assigned) and its type.

Selection of tables in the list is based on table type or a combination of table types. Selected tables can be listed in order of table name or number.

Description

The output produced by the DB2 LIST TABLES command shows, for each DB2 table selected:

- The number of the table in the WBDA
- The columns comprising the primary key of the table
- The table name, if one has been assigned
- The type of table, that is, one of the following:
  - PARENT/ROOT
  - DEPENDENT/PARENT
  - DEPENDENT/LEAF
  - INDEPENDENT,

where the above types are defined as indicated in the following paragraphs.

In DB2, a foreign key relationship is directed from (the primary key of) a parent table to (a foreign key in) a dependent table. If the relationship is derived from a domain association (in the corresponding relational schema), then the foreign key is the primary key of the dependent table. Otherwise, the foreign key is identical to the primary key of the parent table and appears in the dependent table as a non-key set of columns (which can include some but not all of the dependent table's primary key).

A PARENT/ROOT table is a table which participates in one or more foreign key relationships as a parent table only. A DEPENDENT/PARENT table is a table which participates in one or more foreign key relationships as a parent table and which also participates in one or more foreign key relationships as a dependent table.

A DEPENDENT/LEAF table is a table which participates in one or more foreign key relationships as a dependent table only. An INDEPENDENT table is a table which participates in no foreign key relationships at all, that is, it is neither a parent nor a dependent table.

The output of this command can help you to decide which tables to specify as seeds in the DB2 PLOT REFERENTIAL-STRUCTURES command.

Example

The following is an example of output from the DB2 LIST TABLES command for the Department model:

Table 15 List of DB2 Tables Held in WBDA

Number	Key	Name and Type	
1	EMPLOYEE-NO DEPARTMENT-NO	DEPARTMENT-MEMBER	DEPENDENT/LEAF
2	DEPARTMENT-NO OFFICE-LOCATION	OFFICE-DEPARTMENT	DEPENDENT/LEAF
3	DEPARTMENT-NO	DEPARTMENT	DEPENDENT/PARENT
4	EMPLOYEE-NO	EMPLOYEE	PARENT/ROOT
5	OFFICE-LOCATION	OFFICE	PARENT/ROOT
6	MANAGER-NO	MANAGER	DEPENDENT/PARENT

Output from the DB2 LIST CYCLES Command

Introduction

The DB2 LIST CYCLES command produces, for every cycle present in the DB2 design generated in the WBDA, a list of the tables appearing in the cycle. Tables in each cycle can be listed in alphanumeric order or in cyclic order, beginning with the table whose WBDA number is the lowest.

Description

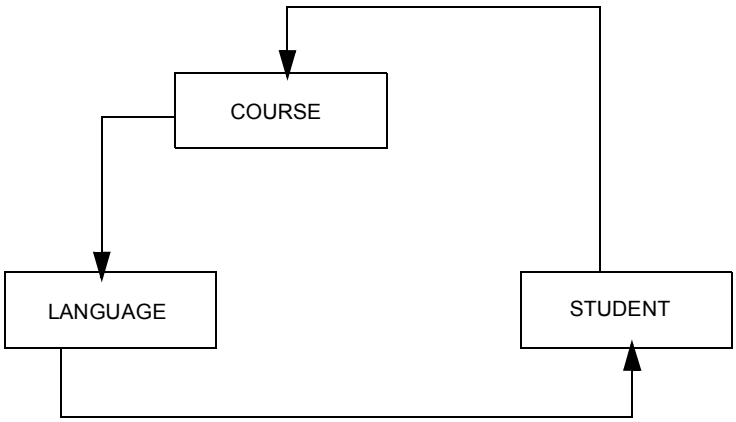
For each cycle found in the DB2 design present in the WBDA, the DB2 LIST CYCLES command produces a list of the tables appearing in the cycle, showing for each table:

- The table WBDA number
- The primary key of the table
- The name of the table, if one has been assigned
- The keyword MULTIPLE, if the table appears in more than one cycle.

Example

In Figure 22, an example is pictured of a cycle with its path of tables and connecting relationships:

Figure 22 • Example Cycle



The output in Table 10 would be produced by the DB2 LIST CYCLES command (in this case, the result is the same whether or not ALPHABETICALLY is specified in the command because cyclic order, beginning with the lowest numbered table, and alphanumeric order happen to be the same):

Table 16 List of DB2 Cycles Held in WBDA

CYCLE

NUMBER	KEY	NAME
1	COURSE-NO	COURSE
2	LANGUAGE-NO	LANGUAGE
3	STUDENT-NO	STUDENT

CYCLE CONTAINS 3 DB2 TABLES

CYCLE

•

•

•

LIST CONTAINS *nn* DB2 CYCLES

Generated DB2 Member Definitions

This section shows the member definitions generated by the DB2 POPULATE and DB2 PREVIEW commands. Each generated DB2 member contains enough information in its clauses for you to use it subsequently to produce SQL CREATE TABLE, INDEX, or VIEW statements. When creating the DB2 objects, DB2 will assign default values to all the remaining clauses. Definitions for the following member types are generated from the tables in the WBDA:

- DB2-TABLE
- DB2-INDEX
- DB2-VIEW
- SYSTEM.

Refer to Chapter 9, "Repository Member Types," on page 341 for the complete syntax of all DB2 member types.

Generated DB2-TABLE Member

Generated DB2-TABLE Definition

What follows is the subset generated by the DB2 PREVIEW or DB2 POPULATE command of the complete DB2-TABLE dictionary member type syntax.

```
>>──ADD──db2-table-name──;──────────────────────────────────────>
>──DB2-TABLE──┬──────────────────────────────────────────────────>
              └──CREATOR-OWNER db2-user──┘
>──COLUMNS──column-specification────────────────────────────────>
>──┬──────────────────────────────────────────────────────────────>
   └──IN tbspace-name──┘
          <<<,<<<<<<<<<<<<<<<<<<<<<<<<<<
>──SEE dataview-name FOR 'SOURCE'──────────────────────────────>
>──;──────────────────────────────────────────────────────────><
```

where *column-specification* is:

```
>──CONTAINS──────────────────────────────────────────────────────>
     <<<<<<<<<<<<<<<<<<<<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
>──column-name──┬──────────────────────────────────────────────────>
                └──NOT-NULL──┘  └──PRIMARY-KEY──┘
>──┬──────────────────────────────────────────────────────────────>
   └──<<<<<<<<<<<<<<,<<<<<<<<<<<<<<
      referential-constraint──┘
```

where *referential-constraint* is:

```
➤──────CONSTRAINT ── FOREIGN KEY──────────────────────────────────➤

      <<<<<<<<<<<<<<<<<<<<<<<<<<<,<<<<<<<<<<<<<<<<<<<<<<<<<
➤──────column-name──┬──────────────────────────────────────────────➤
                    └──MEMBER referred-member──┘
➤──────REFERENCES table-name──────────────────────────────────────➤
```

Example of Generated DB2-TABLE Dictionary Member

The following DB2-TABLE dictionary member is generated for a table called DEPARTMENT, in the WBDA. The table was generated from an input entity called DEPARTMENT-ENT.

The table DEPARTMENT is a dependent table of the table MANAGER. It has columns DEPARTMENT-NO (its primary key), DEPARTMENT-NAME, and MANAGER-NO (its foreign key). The name of the table space assigned to it in the DB2 PREVIEW or DB2 POPULATE command is DEP-TBSP. The name assigned for the CREATOR-OWNER is USER1.

```
ADD DEPARTMENT ;
DB2-TABLE
CREATOR-OWNER USER1
COLUMNS CONTAINS DEPARTMENT-NO NOT-NULL PRIMARY-KEY
                 ,DEPARTMENT-NAME
                 ,MANAGER-NO
CONSTRAINT FOREIGN-KEY MANAGER-NO
         REFERENCES MANAGER
IN DEP-TBSP
SEE DEPARTMENT-ENT FOR 'SOURCE'
;
```

Generated DB2-INDEX Member

Generated DB2-INDEX Definition

What follows is the subset generated by the DB2 PREVIEW or DB2 POPULATE command of the complete DB2-INDEX dictionary member type syntax.

```
➤➤──────ADD db2-index-name── ; ──────────────────────────────────➤
➤──────DB2-INDEX──────────────────────────────────────────────────➤
➤──────┬──────────────────────────────────────────────────────────➤
       └──CREATOR-OWNER db2-user──┘
➤──────UNIQUE ON table-name───────────────────────────────────────➤
➤──────CONTAINS──┬──<,<<<<<<<──────────────────────────────────────➤
                 └──column-name
➤──────┬──<<<,<<<<<<<<<<<<<<<<<<
```

where *db2-user*, *column-name*, *table-name*, and *dataview-name* are valid dictionary member names.

Example of Generated DB2-INDEX Dictionary Member

The following DB2-INDEX dictionary member is generated for a table called DEPARTMENT, in the WBDA. The table was generated from an input entity called DEPARTMENT-ENT.

The table DEPARTMENT has a primary key column called DEPARTMENT-NO. The DB2-INDEX name is constructed by concatenating the table name with the default suffix '-IND'. The name of the CREATOR-OWNER, assigned in the DB2 PREVIEW or DB2 POPULATE command, is USER 1.

```
ADD DEPARTMENT-IND ;
DB2-INDEX
CREATOR-OWNER USER1
UNIQUE ON DEPARTMENT
CONTAINS DEPARTMENT-NO
SEE DEPARTMENT-ENT FOR 'SOURCE'
;
```

Generated DB2-VIEW Member

Generated DB2-VIEW Definition

What follows is the subset generated by the DB2 PREVIEW or DB2 POPULATE command of the complete DB2-VIEW dictionary member type syntax.

```
>>──ADD──db2-view-name──;──────────────────────────────────>
>──DB2-VIEW──────────────────────────────────────────────────>
>──┬──────────────────────────────────────────────────────────>
  └─CREATOR-OWNER db2-user─┘
                 <<<,<<<<<<
>──CONTAINS column-name──────────────────────────────────────>
>──SELECT ALL────────────────────────────────────────────────>
>──FROM table-name───────────────────────────────────────────>
                <<,<<<<<<<<<<<<<<<<<<<<
>──SEE dataview-view FOR 'SOURCE'────────────────────────────>
>──;─────────────────────────────────────────────────────────><
```

where *db2-user*, *column-name*, *table-name*, and *dataview-name* are valid dictionary member names.

Example of Generated DB2-VIEW Dictionary Member

The following DB2-VIEW dictionary member is generated for the table called DEPARTMENT, in the WBDA. The table was generated from an input entity called DEPARTMENT-ENT.

The table DEPARTMENT has columns called DEPARTMENT-NO, DEPARTMENT-NAME, and MANAGER-NO.

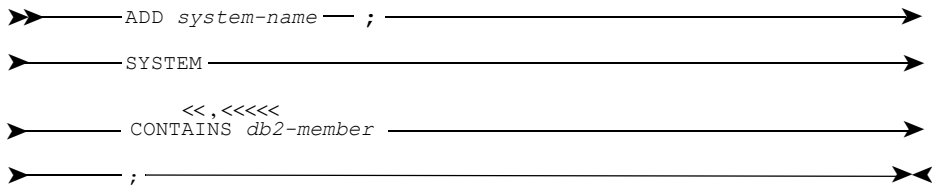
The name of the DB2-VIEW member is constructed from the table name concatenated with the default suffix -VIEW. The name of the CREATOR-OWNER, assigned in the DB2 PREVIEW or DB2 POPULATE command, is USER 1.

```
ADD DEPARTMENT-VIEW;
DB2-VIEW
CREATOR-OWNER USER1
CONTAINS DEPARTMENT-NO, DEPARTMENT-NAME, MANAGER-NO
SELECT ALL
FROM DEPARTMENT
SEE DEPARTMENT-ENT FOR 'SOURCE'
;
```

Generated SYSTEM Member

Generated SYSTEM Definition

What follows is the subset generated by the DB2 PREVIEW or DB2 POPULATE command of the complete SYSTEM dictionary member type syntax.



where *db2-member* is a valid dictionary member name.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of the complete SYSTEM member type syntax.

Example of Generated SYSTEM Dictionary Member

The following SYSTEM dictionary member is generated for a DB2 table called DEPARTMENT, in the WBDA; INDEXES and VIEWS have also been specified for this table in the DB2 PREVIEW or DB2 POPULATE command. The name of the SYSTEM has been specified as DB2-SYSTEM-TEST.

```
ADD DB2-SYSTEM-TEST;
SYSTEM
CONTAINS DEPARTMENT-NO, DEPARTMENT-IND, DEPARTMENT-VIEW
;
```

4 Repository Definition

This chapter includes these sections:

Introduction to Documenting a DB2 DBMS 91

Documenting DB2 Objects 92

Documenting the Columns of Indexes, Tables, and Views 94

Documenting DB2 Security Information 96

Naming Conventions for DB2 Members 97
Generating External Names 97
Naming Guidelines 101

Interrogating Your DB2 Dictionary Schema 102

Introduction to Documenting a DB2 DBMS

Several DB2 member types are available to enable you to document DB2 objects on the repository. Generally, there is a one-to-one correspondence between a DB2 object and the member type used to document it.

If you have more than one relational environment containing shared data, you can fully document the data once in the repository, then define relationships between member definitions to share that documentation. ITEM and GROUP members that document the columns of DB2 tables and views may also be used in other applications in an installation, and in other database schemas.

You can reflect different stages in the life cycle of a DB2 database using a hierarchy of statuses, available with the change management facility. The DB2 catalog cannot do this, since it records only the current status of the DB2 environment. Therefore, the repository can be an important tool for change control in a DB2 environment.

Populate your repository with DB2-TABLE, DB2-VIEW, and DB2-INDEX members using data modeling and design functions. The relational schema in the WBDA is used to generate members for the first-cut DB2 design. These members constitute a first-cut repository schema, which you can go on to develop and complete in the repository.

Refer to Appendix B, "Documenting Other Relational Databases," on page 515 for details of sharing data from several relational databases.

Refer to *ASG-Manager Products Status Concepts* for details of statuses.

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of populating the repository from the WBDA.

Documenting DB2 Objects

DB2 objects in the DB2 database schema are documented in the repository as members of an equivalent type. The table below shows the correspondence between DB2 objects and the repository member types that are used to document them. A DB2 repository member can be said to represent the corresponding DB2 object.

Table 17 Correspondence between Repository Members and DB2 Objects:

Repository Member	DB2 Object
DB2-DATABASE	database
DB2-STOGROUP	storage group
DB2-TBSPACE	table space
DB2-INDEX	index
DB2-TABLE	table
DB2-ALIAS	alias
DB2-VIEW	view
DB2-PROCEDURE	procedure
DB2-TRIGGER	trigger
ITEM (or GROUP)	column or distinct type
DB2-PLAN	plan
DB2-PACKAGE	package
DB2-COLLECTION	collection
DB2-DMS	data manipulation statement
DB2-AUXTABLE	auxiliary table
DB2-TEMPTABLE	temporary table
DB2-FUNCTION	function

Additional member types that do not represent DB2 objects are:

- DB2-LOCATION, which is used to document and generate the location name of a table or view in distributed database environments
- DB2-USER, which is used to document the authorization ID of a DB2-USER
- DB2-PRIVILEGE, which is used to document the privileges held by users.

DB2-USER and DB2-PRIVILEGE members also model the DB2 security system and are used to generate DB2 access privileges as SQL GRANT and REVOKE statements.

Refer to "Naming Conventions for DB2 Members" on page 97 for details of documenting the DB2 security system.

Refer to Appendix C, "Defining and Generating DB2 Member Types," on page 519 for a table of relationships between DB2 member types.

Where possible, consistency is maintained between the syntax of the DB2 member type definitions and the syntax of the corresponding SQL statements required to create the DB2 objects. In most instances, the keywords in a member definition statement are identical to the equivalent DB2 keywords, and they have the same meanings.

To avoid duplicating effort and information when all or part of a member definition is the same as that of another member, we have created a mechanism whereby two or more members can share a definition. The AS clause allows you to refer to some of the clauses in one member's definition from another member, so that those parts of the definition that are common to more than one member, are entered only once in the repository, and only one member needs to be maintained.

Members named in the AS clause usually have the same member type as the member you are defining. For example a DB2-VIEW is defined AS another DB2-VIEW. However, a DB2 member type can refer to a user defined member type based on itself, and vice versa. This allows you to define members representing a relational database other than DB2 and still share data across different relational environments.

If you have SQL/DS and DB2 support you can name the corresponding SQL member type in the AS clause of come DB2 member types. These are shown in Appendix C, "Relationship Between DB2 Member Types" on page 519.

Note:

The only clause that must be present in a member definition for it to encode successfully is the member type identifier. This allows you to document a database schema with as much, or as little, information in each object definition, as is necessary at any stage in the development of the database schema. Definitions can be built up in an incremental top down approach, while giving you full interrogation and reporting capabilities.

When you encode a member, it is checked to ensure that any members referred to are of the correct type. When you generate SQL statements from a member, the generated external name is checked to ensure that it conforms to the requirements of the target external environment. The checks that are made on each clause are indicated in the documentation of each member type.

Certain clauses must be present in a DB2 member definition when you generate an SQL statement from it for generation to be successful. The clauses that must be present for the successful generation of a particular SQL statement are indicated in the documentation of each member type.

Refer to Appendix B, "Documenting Other Relational Databases," on page 515 for details of using DB2 member types to define another relational database.

Refer to Appendix C, "Relationship Between DB2 Member Types" on page 519, for a table showing relationships between DB2 member types.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of the AS clause.

Documenting the Columns of Indexes, Tables, and Views

You can document the types of data that can be held in the corresponding columns of DB2 tables and views, using the form description in ITEM members. These ITEMs are referred to by DB2-TABLE, DB2-VIEW, DB2-INDEX, and DB2-PRIVILEGE members. Since DB2 objects can contain many columns, DB2 members types may refer to many ITEM members. To simplify the references, you can define a number of ITEM members as belonging to a GROUP, which the DB2 member then refers to.

When you generate SQL statements and host language structures using either the export to DB2 panels or the DB2 CREATE, DB2 DECLARE, DB2 ALTER, DB2 PRODUCE, and DB2 GRANT commands, the data type is generated from the form description.

The data type of a generated column is determined firstly by the form keyword specified in the COLUMNS clause in a DB2-INDEX, DB2-TABLE, or DB2-VIEW member definition, and secondly by the VERSION number specified in the member definition of the ITEM, which represents the column. The form keyword specified in the COLUMNS clause applies to all the ITEMs and GROUPs specified in the CONTAINS clause. If the specified form of any individual ITEM does not hold the correct data type, you can specify for that ITEM an additional version, which does hold the correct data type.

The data type generated for host language data structures also depends on the target language.

Refer to Appendix C, "Defining and Generating DB2 Member Types," on page 519 for details of the data types generated from different form descriptions.

When PL/1, COBOL, or Assembler host language data structures are generated from an ITEM that includes any of these USAGE clauses: USAGE TIME, USAGE DATE, and USAGE TIMESTAMP the form description is ignored. However, you should still define a form description, in order to ensure that the generated length of the field is compatible with environments other than DB2.

In ITEMs that include a USAGE TIME or USAGE DATE clause, the form description should be specified as a CHARACTER field with a length compatible with the default length specified in the global variables MPDY_CM_LOCAL_TIME and MPDY_CM_LOCAL_DATE.

In an ITEM that includes a USAGE TIMESTAMP clause, the form description should be specified as a CHARACTER field with a length of 26.

The DB2 data type of a column generated from a GROUP that is not expanded depends on the aggregate length and whether any of the contained ITEMs are of variable length. It is one of these:

CHAR(*m*)	Where *m* is the aggregate length of fixed fields that constitute the group and *m* is less than 255 characters.
VARCHAR(*m*)	Where *m* is the aggregate length of fields that constitute the group and one or more of the fields are of variable length and *m* is less than 255 characters.
LONG VARCHAR(*m*)	Where *m* is the aggregate length of fields that constitute the group (regardless of whether none or some are of variable length) and *m* is greater than or equal to 255.

To find the length of the fields, produce a record layout from a GROUP member using either the export panels or the PRODUCE command.

If the aggregate length of the record is less than 255 and any of the fields in the record layout are marked as VARIABLE, the DB2 data type is VARCHAR(*m*), and if no fields are marked as VARIABLE, the DB2 data type is CHAR(*m*). If *m* is equal to or greater than 255, the DB2 data type is always LONG VARCHAR.

Note:

The data types of generated columns are expressed differently according to the destination host language (SQL, PL/1, COBOL, or Assembler). Refer to Appendix C, "Data Types Generated from Form Descriptions" on page 522, for a table that shows how column data types for different host languages are generated from form descriptions in ITEM members.

Refer to Appendix C, "Defining and Generating DB2 Member Types," on page 519 for details of data types generated for columns.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for syntax of the ITEM member type.

Documenting DB2 Security Information

DB2's security system enables you to control access to objects in the DB2 environment. The DB2-USER and DB2-PRIVILEGE member types are provided in the repository schema to document the DB2 security system.

A DB2-user must have an authorization ID to be able to sign on to DB2. This authorization ID is recorded in the repository as a DB2-USER member type.

DB2 privileges are granted to users by means of an SQL GRANT statement. All privileges are granted by a particular user (the GRANTOR) to another user (the GRANTEE). The privilege is recorded in the DB2 catalog, and is used by DB2 whenever it is necessary to check if the signed on user has permission to perform an particular DB2 operation. You can document privileges in the repository as DB2-PRIVILEGE members.

Together, the DB2-USER and DB2-PRIVILEGE member types allow you to document the DB2 objects to which particular users have access, and what access rights they have. Database Administrators interrogate the repository model of the DB2 security system using repository functions or repository interrogation commands.

The DB2-USER and DB2-PRIVILEGE repository definitions are also used to generate SQL GRANT, SQL REVOKE, and SQL CREATE SYNONYM statements.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of defining DB2-PRIVILEGE and DB2-USER members.

Naming Conventions for DB2 Members

Generating External Names

When you generate SQL statements from DB2 members the external names of the DB2 objects that they represent must conform to the naming rules for DB2. In this case, the external name of an object is the name by which it is known to DB2 and which is recorded in the DB2 catalog, that is, the "DB2 name."

In addition, when you generate COBOL, PL/1, or Assembler host language structures from a DB2 member containing columns, the external names of the columns must also conform to the rules appropriate for the relevant host language. In this case, the external name of an object is the name by which it is known by COBOL, PL/1, or Assembler.

In order to generate an acceptable external name, a number of processes are applied. Therefore, when you document a DB2 member in the repository, it is important to know how the external name is derived from the repository definition.

If you want to generate external names from the ALIAS clause in member definitions, your systems administrator must ensure that your DB2 profile is tailored appropriately, and that an alias table exists for each language you want to generate. You can change the name to be generated using the name editing options available in the export panels and commands.

Hyphens separating the constituent parts of names are changed to underscores, on generation of an SQL statement.

All derived external names are subjected to a final check, on generation, to ensure they are valid for the relevant external environment. The check includes ensuring that external names are not longer than permitted by the relevant external environment. If a name is too long, then some characters are removed to reduce the length.

See Chapter 5, "Export to DB2," on page 105 for details of DB2 profiles.

See Chapter 8, "Commands," on page 175 for details of name editing options.

See Appendix A, "Name Reduction Process," on page 511 for details of how names are reduced.

Generating Column Names

The names of columns in tables, views, and indexes are derived from the first of the following sources that can be found.

- The name specified in the COLUMN NAME clause of a DB2-VIEW member
- The name specified in the KNOWN AS clause of the DB2-TABLE, DB2-VIEW, or DB2-INDEX in which the columns are defined
- The name defined in the ALIAS clause of the ITEM or GROUP member that represents the column, when your DB2 profile is tailored to use ALIAS clauses
- The repository name of the ITEM or GROUP member that represents the column.

The derived column name is then:

- Edited according to any name editing options specified in the export panel or the DB2 command
- Reduced to 18 characters, if the name exceeds this maximum
- Suffixed with an underscore and integer, if the generated name duplicates another.

The unqualified DB2 name of a column may be no longer than 18 characters. Refer to Appendix A, "Name Reduction Process," on page 511 for details of the name reduction process.

Generating Names for Aliases, Tables, Databases, Indexes, Views, Table Spaces, and Storage Groups

The names of aliases, tables, views, indexes, databases, table spaces, and storage groups are derived from the first of the following sources than can be found:

- The name specified in the ALIAS clause of the member that represents the object, when your DB2 profile is tailored to use ALIAS clauses
- The repository name of the member that represents the object.

The derived object name is then:

- Edited according to any name editing options specified in the export panel or the DB2 command
- Reduced to the required number of characters, if the name exceeds the maximum.

Note:

Alias, index, table, table space, and view names must have a qualifier defined to be generated successfully.

Databases and storage groups have unqualified names of no more than eight characters.

Table spaces have names of no more than eight characters, qualified with a database name.

Aliases, indexes, tables, and views have names of no more than 18 characters, qualified with an owner name.

Tables and views can optionally have three-part names, with both location and owner qualifiers.

Generating Names for Synonyms, Catalogs, Passwords, and Programs

The DB2 object names of synonyms, catalogs, passwords, and programs are derived directly from the corresponding names specified in the repository definitions where they appear. The name of a synonym may be no longer than 18 characters. The names of catalogs, passwords, and programs may be no longer than eight characters.

Generating Two-part Names

The default setting in your DB2 profile is for two-part name generation.

Table space names can be qualified with the name of the database in which the table space resides. The database qualifier is derived from the first of the following sources that can be found:

- The name specified in the ALIAS clause of the DB2-DATABASE member named in the IN clause of the DB2-TBSPACE being generated, when your DB2 profile is tailored to use ALIAS clauses
- The member name of the DB2-DATABASE named in the IN clause of the DB2-TBSPACE being generated.

Alias, index, table, and view names can be qualified with the name of the user who owns them.

The user qualifier is derived from the first of these sources that can be found:

- The name specified in the SQLID clause of an export panel or a DB2 command
- The name specified in the ALIAS clause of the DB2-USER member named in the CREATOR-OWNER clause of the member being generated, when your profile is tailored to use ALIAS clauses
- The repository name of the DB2-USER named in the CREATOR-OWNER clause of the member being generated.

The generated user qualification corresponds to the DB2 authorization ID, and may be no longer than eight characters.

To generate alias, index, table, and view names with no owner qualifier, specify the SQLID clause in the DB2 command with a single blank space, as follows:

```
SQLID ' '
```

If no user qualifier can be found, the name is generated with a prefix of eight question marks, indicating that the qualifier is missing. You should correct the CREATOR-OWNER clause in the member definition, or specify an SQLID clause in the DB2 command, and regenerate the SQL statement.

The object you are generating may refer to other objects. Referenced object names are generated with qualifiers. If you do not want referenced object names to have qualifiers you can tailor your DB2 profile.

See Chapter 5, "Export to DB2," on page 105 for details of tailoring your DB2 profile.

Generating Three-part Names

If your profile is set to three-part name generation, an additional location qualifier is generated for tables and views.

The location qualifier is derived from the first of these sources that can be found:

- The name specified in the LOCATION clause of an export panel or a DB2 command
- The name specified in the ALIAS clause of the DB2-LOCATION member named in the LOCATION clause of the DB2-USER member named in the CREATOR-OWNER clause of the DB2-TABLE or DB2-VIEW member being generated where your profile is set to select ALIAS clauses
- The repository name of the DB2-LOCATION referred to in the LOCATION clause of the DB2-USER member named in the CREATOR-OWNER clause of the DB2-TABLE or DB2-VIEW member.

If no location qualifier can be found, the name is generated with a prefix of 16 question marks, indicating that the qualifier is missing. You should correct the member definition, or specify a LOCATION clause in the DB2 command, and regenerate the SQL statement.

Naming Guidelines

Although mechanisms are provided to derive names for DB2 objects that are acceptable when DB2 member types are used in the generation of SQL statements or host language data structures, you should be aware that, in many cases, arriving at an acceptable name may involve some loss of information, that is, you will not know the DB2 name of an object by looking at the member in which it is documented. This is the case when. name editing or name reduction takes place and when you specify names in the KNOWN AS clauses of DB2-TABLE, DB2-VIEW, DB2-PRIVILEGE, DB2-USER, and DB2-INDEX member definitions. When such a loss of information does take place, it may be no longer possible to determine unambiguously the repository origin of an external name. It is important, therefore, to establish a naming strategy before you start to define DB2 members in your repository.

The simplest approach to a reliable naming strategy is to ensure that all external names of DB2 objects are derived directly from the repository member name of the member that represents it in the repository.

The way to achieve this is to omit these clauses in your member definitions:

- The COLUMN NAME clause of a DB2-VIEW member
- The KNOWN AS clause of DB2-TABLE, DB2-VIEW, DB2-PRIVILEGE, DB2-USER, and DB2-INDEX members
- The ALIAS clause of any member (assuming your environment is tailored to generate aliases as external names)

and to omit these clauses from export panels or DB2 commands that generate SQL statements:

- The LOCATION clause
- The SQLID clause
- Name editing options

Since the external name is taken from the repository name of the member that represents a DB2 object, there is a direct correspondence between the repository and external environments. Therefore, it is straightforward to match DB2 objects with the repository members that represent them, since they have the same name in the repository as they do on the DB2 catalog, and in host language structures where they are used.

This is the simplest approach and the most efficient in terms of processing time.

However, this is not practical in all environments. For example, you may wish to include in your DB2 schema an already existing GROUP or ITEM with a name the length of which exceeds the allowed maximum for the target external environment. This situation may arise when, for instance, the external environment is an Assembler language where the maximum length of names is eight characters.

In this situation, consider using aliases that correspond to the external environment in question. If, for example, you use SQL and COBOL, set up SQL and COBOL aliases for each member. Although aliases are not checked to ensure their uniqueness in the repository (which means that two unrelated members could have the same alias), interrogation commands that can detect a duplicate alias can be used to guard against alias duplication.

For example, you can interrogate the repository to find out all the members of the type DB2-TABLE that have a specific alias name. You can then change the alias names of all but one of these DB2-TABLE members, in order to achieve unique names. The generated table name in the DB2 catalog then corresponds with the SQL ALIAS of one and only one DB2-TABLE repository member.

Note:

The extra work involved in deriving external names from aliases can increase processing time. By deactivating the alias tables you can reduce processing.

If you use COBOL, a more descriptive variant is possible, since COBOL allows names of up to 30 characters in length, while SQL allows a maximum of 18 characters. For example, the COBOL name of the table could be EMPLOYEE MAIN TABLE (which is 19 characters long).

Choose repository and external names carefully. If necessary, use aliases and interrogation capabilities to implement a sound naming strategy that allows for a direct correspondence between your DB2 repository schema, the database schema which it represents, and other external environments in which objects may be used.

Interrogating Your DB2 Dictionary Schema

Member type keywords are available for use in these commands:

- BULK
- GLOSSARY
- LIST
- PERFORM
- REPORT
- WHICH

These interrogation keywords are added to the member type keywords so that the definitions of DB2 objects may be processed in the repository in the same way as other repository members:

- DB2-ALIAS
- DB2-AUXTABLE
- DB2-COLLECTION
- DB2-DATABASE
- DB2-DMS
- DB2-INDEX
- DB2-FUNCTION
- DB2-LOCATION
- DB2-PACKAGE
- DB2-PLAN
- DB2-PRIVILEGE
- DB2-PROCEDURE
- DB2-STOGROUP
- DB2-TABLE
- DB2-TBSPACE
- DB2-TEMPTABLE
- DB2-TRIGGER
- DB2-USER
- DB2-VIEW

In addition, the alias type keyword SQL is available in the ALIAS clause. Defining an SQL ALIAS for a DB2 member, ensures that the name defined is used in the DB2 environment.

5 Export to DB2

This chapter includes these sections:

Generating Output 106
Submitting Generated Output to Your Relational Environment 107

Tailoring Output 109
Introduction to Tailoring 110
Generating Object Names and External Names from Aliases 114
Tailoring DATE and TIME Character Field Lengths 115
Generating a Host Language Data Structure with an SQL DECLARE Statement 116
Generating an SQL DECLARE Statement with a Host Language Data Structure 117
Setting the Release of DB2 117
Setting the Release Flag 117
Generating Flat or Nested Data Structures 117
Generating Indicator Structures 118
Generating Indicator Suffixes on Structures 118
Setting Suffixes Applied to Indicator Array Names 119
Setting Suffixes Applied to Variable-Length Column Names 119
Automatically Generating SQL COMMENT ON/LABEL ON Statements 119
Generating One-, Two-, or Three-part Names for DB2 Objects 120
Setting a Tolerance Level for Output 121
Setting the SQL Escape Character 121
Setting Width of Output for SQL COMMENT ON Statements 122
Setting Width/Indent of the SQL DROP Impact Analysis Report 122
Allowing an Optional Space Character when Generating SQL DECIMAL Datatypes 122
Accessing a Specific DB2 Subsystem or Plan 123
Setting EXPORT Generated Object-name Length 123
Setting the Generated Column Data Type 124
Creating an INSERT Statement for Stored Procedures 124
Introduction to User Exits 124
Taking User Exits when Accessing a Repository Member 125
Taking User Exits For Specified DB2 Export Functions 125
Taking User Exits for an Individual Export Function 129
Setting Switches for Non-standard SQL Generation 130

Generating Output

Export to DB2 functions are provided by Manager Products DB2 commands and export panels (menu E310000). Using these functions, you can:

- Generate these SQL statements:

ALTER	GRANT
COMMENT ON	LABEL ON
CREATE	REVOKE
DECLARE	CREATE SYNONYM
DROP	DROPSYNONYM

- Generate Assembler, COBOL, or PLl host language data structures, for use (with embedded SQL statements) within application programs
- Generate reports describing the layout of repository members documenting DB2 tables or views
- Generate BIND or REBIND subcommands to submit to your DB2 environment
- Produce a report showing the impact of a specific SQL DROP statement in your DB2 environment
- Produce an estimate of the size of an index, or a table and its rows, and so estimate storage space before generating an SQL CREATE statement
- Set up a trace facility displaying diagnostic information when generating SQL statements or host language data structures.

DB2 definition functions provide the repository members needed to document the objects which exist or are to be created in your DB2 environment.

Column data types in tables or views are given in ITEM or GROUP members referred to (in the CONTAINS attribute) by DB2-TABLE and DB2-VIEW members from which output is generated.

See "Documenting DB2 Security Information" on page 96 for details of how the data type of columns are derived.

You can automatically file generated output in a USER-MEMBER, or in an external file, which you can use as input to your DB2 environment. You can also use dynamic SQL functions to dynamically submit the SQL statements you have generated to your DB2 environment from within Manager Products.

See Chapter 6, "Dynamic Import/Export," on page 131 for details of dynamic SQL Functions.

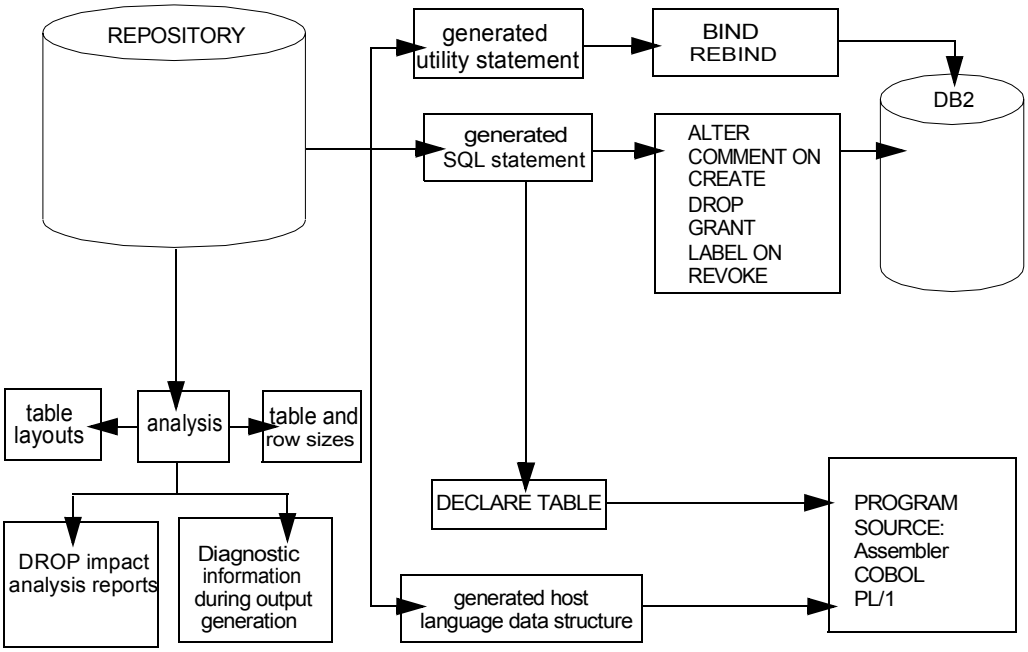
You can tailor the output generated by altering the DB2 profile.

See "Tailoring Output" on page 109 for details of tailoring output.

You can use these functions to document, implement, and maintain your DB2 environment.

Figure 23 illustrates the process of exporting generated output.

Figure 23 • Exporting Generated Output



Submitting Generated Output to Your Relational Environment

Using dynamic SQL functions, you can dynamically submit generated SQL statements to your relational environment, from within Manager Products. To use dynamic SQL functions you must have either DB2 or SQL/DS available on the same CPU and operating system as Manager Products.

Alternatively, you can file generated output either in a USER-MEMBER on the MP-AID, or in an external file. If you file output on the MP-AID, you can then later choose to transfer the output from the USER-MEMBER into an external file, using the TRANSFER command. You can use the external file as input to your relational environment, or for inclusion in your application programs.

Dynamic SQL functions can only submit SQL statements that can be dynamically prepared for execution. You therefore cannot submit SQL DECLARE statements using dynamic SQL functions.

SQL DECLARE statements and host language data structures should be embedded in application programs and must be output to an external file either directly or indirectly (as described above). Your application program can then use SQL COPY or INCLUDE statements to reference the external file.

See Chapter 6, "Dynamic Import/Export," on page 131 for details of dynamic SQL functions. See "Output Generation Options" on page 334 for details of filing generated output.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of the TRANSFER command.

Tailoring Output

Figure 24 • Structure of the Example DB2 Profile

Logon/Global Profile

Instructions
executive-routine;

executive-routine (MPDY12PROF)

```
mpxx literal=:
RETAIN
  global mpdy_gl_profilename
  command mpdy_db2_release_no /* unset = latest DB2 release (3.1) */
  command mpdy_db2_release_flag /* unset = Error message issued. */
  mpdy_gl_profilename  = :MPDY12PROF: /* name of this DB2 profile
  mpdy_db2_release_no  = :6.0:
  mpdy_db2_release_flag = :W:
  mpdy_cm_db2alias  = :MPR-EX-DB2:
  mpdy_cm_sqlalias  = :MPR-EX-DB2:
  mpdy_cm_cobolalias  = :MPR-EX-COB:
  mpdy_cm_plialias  = :MPR-EX-PL1:
  mpdy_cm_balalias  = :MPR-EX-BAL:
mpr :MPDYADDXIT OUT DB2-CREATE-TABLEMPDY12OXCR ;:
mpr :MPDYADDXIT COL DB2-CREATE-TABLEMPDY12DXCL ;:
mpr :MPDYADDXIT OBJ DB2-CREATE-TABLEMPDY12DXOB ;:
mpr :MPDYADDXIT OUT DB2-CREATE-VIEW MPDY12OXCV ;:
mpr :MPDYADDXIT WBTA DB2-CREATE-TABLE MPDY12DXWB ;:
mpr :MPDYADDXIT OUT DB2-LABEL-TABLEMPDY12OXCR ;:
mpr :MPDYADDXIT COL DB2-LABEL-TABLEMPDY12OXCR ;:
mpr :MPDYADDXIT OBJ DB2-LABEL-TABLEMPDY12OXCR ;:
mpr :MPDYADDXIT WBTA DB2-LABEL-TABLEMPDY12OXCR ;:
mpr :MPDYADDXIT OUT DB2-PRODTAB-COB-FOR-WS MPIDY1 2OXPC ;:
mpr :MPDYADDXIT OUT DB2-PRODTAB-PL1-FOR-WS PDYI2OXPP ;:
mpr :MPDYADDXIT OUT DB2-PRODTAB-BAL-FOR-WS PDY12OXPB ;:
  global mpdy_db2ssn mpdy_planname
/*mpdy_db2ssn = :xxxxxx:
  mpdy_planname = :MPDB2:   /* ASG default
```

Introduction to Tailoring

You can create a DB2 profile, which can tailor generated output. For example, use a DB2 profile to derive DB2 object names in SQL statements from aliases instead of member names.

The DB2 profile is called from either the Global or Logon profile, and is either:

- A Corporate executive routine, created by the systems administrator for all users, or
- A User executive routine, created by individual users, and applying to these users only. These profiles should be created under the supervision of the systems administrator, as they override DB2 profiles which are Corporate executive routines.

ASG provides these example EXECUTIVE members:

- MPDY12PROF as a DB2 profile
- MPR-EX-DB2, MPR-EX-COB, and MPR-EX-BAL, as DB2 alias tables
- MPDY12OXCR, MPDY12OXPB, MPDY12OXPC, MPDY12OXPP, and MPDYl2OXCV as user exit routines

These members are shown in Figure 24 on page 109.

You can call user exit routines by invoking the COMMAND member MPDYADDXIT in your DB2 profile.

Refer to the sections "Setting EXPORT Generated Object-name Length" on page 123 through "Taking User Exits when Accessing a Repository Member" on page 125 for details of calling user exit routines.

You can tailor the report showing the impact of an SQL DROP statement by altering the Corporate executive routine MPDYl2DRRL. We recommend that you consult your local ASG product supplier before altering this routine.

You can alter the values of these variables to tailor output to your DB2 environment:

Variable Name	Purpose
MPDY_GL_PROFILENAME	Mandatory global variable: must be declared and set to the DB2 profile name.
MPDY_CM_DB2ALIAS	Derives DB2 object names in SQL statements from aliases.
MPDY_CM_BALALIAS MPDY_CM_COBOLALIAS MPDY_CM_PL1ALIAS	Derives external names in Assembler, COBOL, and PLI host language data structures from aliases.

Variable Name	Purpose
MPDY_CM_ALIAS(*n*)	Specifies the type of alias from which you want to derive DB2 object names and external names.
MPDY_CM_ALIASNUM	Stops DB2 object names and external names being derived from aliases.
MPDY_CM_LOCAL_DATE MPDY_CM_LOCAL_TIME	Generates column variables in host language data structures that are compatible with your DB2 installation settings for date and time.
MPDY_CM_HOSTOPT	Automatically generates a host language data structure when generating an SQL DECLARE statement.
MPDY_CM_DECLAREOPT	Automatically generates an SQL DECLARE statement when generating a host language data structure.
MPDY_DB_SYSTEM	Sets the specific relational DB system.
MPDY_DB_SYSTEM_VARIANT	Specific variant for which commands should be done.
MPDY_DB2_RELEASE_NO	Specifies the release of DB2 in use.
MPDY_DB2_RELEASE_FLAG	Specifies the release flag.
MPDY_ORACLE_RELEASE_NO	Specifies the release of Oracle in use.
MPDY_ORACLE_RELEASE_FLAG	Specifies the release flag.
MPDY_INFORMIX_RELEASE_NO	Specifies the release of Informix in use.
MPDY_INFORMIX_RELEASE_FLAG	Specifies the release flag.
MPDY_SYBASE_RELEASE_NO	Specifies the release of Sybase in use.
MPDY_SYBASE_RELEASE_FLAG	Specifies the release flag.
MPDY_FOR_OPTION	Specifies what type of host language data structure is generated: two-level or multi-level.
MPDY_INDICATOR_ONLY	No host structure.
MPDY_INDICATOR_HDR1	No host structure.
MPDY_INDICATOR_HDR2	No host structure.
MPDY_INDICATOR_PREFIX	Prefix for variables.
MPDY_INDICATOR_OPTION	Generates indicator structures when generating host language data structures.
MPDY_INDICATOR_SUFFIX	Specifies a suffix for indicator variables.

Variable Name	Purpose
MPDY_IND_ARRAY_SUFF_*n*	Specifies a suffix for indicator arrays.
MPDY_INDICATOR_DELBAL	Deliverable for Assembler.
MPDY_INDICATOR_DELPLI	Deliverable for PL/I.
MPDY_INDICATOR_DELCOB	Deliverable for COBOL.
MPDY_COB_SUFF_*n*	Specifies a suffix for the name generated when a column has a variable length definition.
MPDY_COB_HOST_SUFF1	Specifies a suffix for COBOL host structures on level 01.
MPDY_COB_HOST_SUFF2	Specifies a suffix for COBOL host structures for subsequent levels.
MPDY_COB_HOST_PREF1	Specifies a prefix for COBOL host structures on level 01.
MPDY_COB_HOST_PREF2	Specifies a prefix for COBOL host structures for subsequent levels.
MPDY_CM_COMMENTOPT MPDY_CM_LABELOPT	Enables SQL COMMENT ON or LABEL ON statements to be automatically generated when generating an SQL CREATE statement.
MPDY_CM_NAME_QUAL	Generates one, two, or three-part names for DB2 aliases, indexes, packages, tables, and views.
MPDY_CM_SENDF_THRESH	Sets tolerance level for sending output.
MPDY_CM_SQL_ESCAPE	Sets SQL escape character.
MPDY_CM_COMMENT_LM MPDY_CM_COMMENT_RM	Sets default left and right margins for COMMENT ON output.
MPDY_CM_DR_WIDTH MPDY_CM_DR_INDENT	Sets default width and indent for the DROP impact analysis report.
MPDY_CM_DCML_SPACE	Generates a space after the comma when generating SQL DECIMAL(*n*,*m*) datatypes.
MPDY_DB2SSN	Specifies a DB2 subsystem name.
MPDY_PLANNAME	Specifies a DB2 plan name.
MPDY_CM_BALLENGTH	Specifies maximum length of names from BAL.
MPDY_CM_COBLENGTH	Specifies maximum length of names from COBOL.

Variable Name	Purpose
MPDY_CM_PL1LENGTH	Specifies maximum length of names from PL1.
MPDY_CM_NUM_GEN	Specifies the data type of the column generated from an item with form-description NUMERIC- CHAR. Valid options are DEC or CHAR. The default is DEC.
MPDY_PRIMARY_IS_NOTNULL	Adds `NOT NULL` if the column is primary key.
These variables are used for DB2 5.*x*:	
MPDY_STOPROC_AUTHID	Specifies the authorization ID
MPDY_STOPROC_LUNAME	Specifies the LU-Name of the system executing the call.
MPDY_STOPROC_LINKAGE	Specifies the linkage conventions of parameters.
MPDY_STOPROC_COLLID	Specifies the name of the collection to be used
MPDY_STOPROC_LANGUAGE	Specifies either Assembler, COBOL, PL\I, or C as the program language.
MPDY_STOPROC_ASUTIME	Specifies the number of service units before forcing termination.
MPDY_STOPROC_STAYRES	Flag to keep the procedure in memory after execution.
MPDY_STOPROC_RUNOPTS	LE/370 run-time-options.
MPDY_STOPROC_WLENV	Specifies the name of the workload manager environment.
MPDY_STOPROC_PGMTYPE	Specifies either MAIN or SUB.
MPDY_STOPROC_EXTSEC	Flag to control if non-SQL-resources need RACF-authorization.
MPDY_STOPROC_COMMIT	Flag to control automatic COMMIT when the procedure ends.
MPDY_CM_OBJEXIT MPDY_CM_COLEXIT	Specifies a user exit to be taken for each DB2 object, or each column in a DB2 table, index, or view, during generation.

Generating Object Names and External Names from Aliases

You can tailor generated output to derive DB2 object names in SQL statements and external names in host language data structures from aliases.

To generate aliases as DB2 object names in SQL statements, enter the following in your DB2 profile:

```
MPDY_CM_DB2ALIAS = :sql-alias-table:
```

To generate aliases as external names in Assembler, COBOL, and PL1 host language data structures, enter the following in your DB2 profile:

```
MPDY_CM_BALALIAS = :assembler-alias-table:
MPDY_CM_COBOLALIAS = cobol-alias-table:
MPDY_CM_PLIALIAS = pli-alias-table:
```

where *sql-alias-table, assembler-alias-table, cobol-alias-table*, and *pli-alias-table* are alias tables: executive routines containing directives specifying the types of alias from which the DB2 object or external name is derived.

You should create a separate alias table for each language. For each table you should specify, in order of preference, the types of alias from which, if present, you want the DB2 object or external names to be derived.

For example, you can derive external names from COBOL aliases when generating COBOL host language data structures, by specifying the COBOL alias type in the cobol-alias-table.

Refer to "Interrogating Your DB2 Dictionary Schema" on page 102 for details of how names are derived.

To specify the type of alias from which you want to derive DB2 object or external names, enter the following in each alias table:

```
MPDY_CM_ALIAS(n) = :alias-type:
```

where *alias-type* is any of the types of aliases available in your repository.

You can specify alternative keywords (alias-type synonyms) by using the CONTROL NEW-ALIAS command. DB2 object names and external names are only generated correctly when you use the first alias-type keyword for any given alias. Use the SHOW ALIAS-TYPES command to find out the types of aliases available in your repository.

Refer to your installation manual for details of the CONTROL NEW-ALIAS command.

where *n* is an integer specifying the order of interrogation of each alias-type to derive the DB2 object or external name, if more than one directive has been entered.

The DB2 object or external name is derived from the first of the interrogated alias-types found in the repository member from which the output is being generated.

Alias-type 1 is interrogated first. If it is not present, alias-type 2 is interrogated next and so on until an interrogated alias-type is found.

If alias-type is NO-TYPE the DB2 object or external name will be derived from the fIrst general alias interrogated. You can only derive the DB2 object or external name from the first general alias in the member.

If a member has no ALIAS attribute or no ALIAS attribute of the types specified in the Alias Table then the default (the member's name) will be generated as the DB2 object or external name.

Manager Products provide four example EXECUTIVE members as alias tables. These are MPR-EX-BAL, MPR-EX-PL1, MPR-EX-COB and MPR-EX-DB2, for the Assembler, PL1, COBOL and SQL languages. You can use these as models.

To return to the default either remove the relevant instructions or enter:

```
MPDY_CM_ALIASNUM = 0
```

in a particular alias table to deactivate that alias table, or in the DB2 profile to deactivate all the alias tables.

Refer to the DB2-DATABASE member type in Chapter 9, "Repository Member Types," on page 341 for an example of an object name generated from an alias.

Tailoring DATE and TIME Character Field Lengths

You can tailor the character field lengths generated for column variables in host language data structures that correspond to a DB2 data type of DATE or TIME. You should set them to the value of the LOCAL DATE LENGTH and LOCAL TIME LENGTH installation options for your DB2 environment. The length values specified also apply in the context of DB2 SIZE and DB2 RECALCULATE.

To tailor DATE lengths, enter the following in your DB2 profile:

```
MPDY_CM_LOCAL_DATE = n
```

where *n* is the character field length. The minimum value, 10, is the default.

To tailor TIME lengths, enter the following in your DB2 profile:

```
MPDY_CM_LOCAL_TIME = n
```

where *n* is the character field length. The minimum value, 8, is the default.

Generating a Host Language Data Structure with an SQL DECLARE Statement

You can automatically generate a host language data structure whenever you generate an SQL DECLARE statement. Both the SQL DECLARE statement and the host structure are generated in the same language.

To automatically generate a host structure with an SQL DECLARE statement, enter the following in your DB2 profile:

```
MPDY_CM_HOSTOPT = :ON:
```

Processing times are faster if the SQL DECLARE statement and the host language data structure are generated together rather than individually.

If you have also tailored your DB2 profile to generate:

- Character field lengths compatible with your DB2 installation settings for time and date, and/or
- A host language indicator structure
- A specific type of host structure (two-level or multi-level)

these alterations are applied to the generated output.

If you have tailored your DB2 profile so that DB2 object names are derived from aliases:

- For an SQL DECLARE statement, the DB2 Alias Table is always used
- For a host language data structure, the Alias Table of the relevant language is used.

Refer to "Generating Object Names and External Names from Aliases" on page 114 for details of deriving object names from aliases.

Names generated for a host language data structure are reduced (when necessary) by the name reduction process to the individual target language requirements.

Refer to Appendix A, "Name Reduction Process," on page 511 for details of the name reduction process.

Note:

External names generated for a host structure may differ from those generated from the same member for an SQL DECLARE statement.

To generate SQL DECLARE statements and host language data structures separately, either remove the variable from the DB2 profile, or alter it to read:

```
MPDY_CM_HOSTOPT = :OFF:
```

Generating an SQL DECLARE Statement with a Host Language Data Structure

To automatically generate an SQL DECLARE statement whenever you generate host language data structures, enter the following in your DB2 profile:

```
MPDY_CM_DECLAREOPT = :ON:
```

Note:

This variable acts in the same way as the WITH-DECLARE keyword in the DB2 PRODUCE command.

Refer to the DB2 PRODUCE command in Chapter 8, "Commands," on page 175 for details of generating host language data structures.

Setting the Release of DB2

To ensure compatibility between DB2 and generated SQL statements, you can set the contents of the variable MPDY_DB2_RELEASE_NO to specify which version of DB2 is installed. To set its contents, enter the following in your DB2 profile:

```
MPDY_DB2_RELEASE_NO = :release:
```

where *release* is 2.2, 2.3, or 3.1. 2.2 represents version 2.2 itself or earlier versions. The default value is 3.1. If unset, the latest version is assumed.

Setting the Release Flag

The release flag controls the action taken if a DB2 version compatibility problem should arise, for example, if a member containing version 2.3 clauses is encountered where the active DB2 release is 2.2. To set the flag, enter the following in your DB2 profile:

```
MPDY_DB2_RELEASE_FLAG = :flag:
```

where *flag* is E, G, W, or S. The default is E. These values have the following meanings:

E: An error message is issued; SQL is not generated.
G: A warning message is issued; SQL is generated.
W: A warning message is issued; SQL is not generated.
S: All messages are suppressed; SQL is not generated.

Generating Flat or Nested Data Structures

You can generate either flat (two-level) or nested (multi-level) host language data structures.

To generate flat data structures (the default option), enter the following in your DB2 profile:

```
MPDY_FOR_OPTION = :SQL:
```

To generate nested data structures, enter the following in your DB2 profile:

```
MPDY_FOR_OPTION = :WS:
```

The default is SQL.

Generating Indicator Structures

You can automatically generate indicator structures when generating corresponding host language data structures.

To generate indicator structures as arrays, enter the following in your DB2 profile:

```
MPDY_INDICATOR_OPTION = :ARRAY:
```

To generate indicator structures as structures which match the main host structure, enter the following in your DB2 profile:

```
MPDY_INDICATOR_OPTION = :STRUCTURE:
```

To return to the default (so that no indicator structures are generated), remove this variable from the DB2 profile.

Note:

You can use the INDICATOR ARRAY and INDICATOR STRUCTURE keywords in the DB2 PRODUCE command to similar effect.

Generating Indicator Suffixes on Structures

You can specify two-character suffixes to be placed after the names of generated indicator variables, when indicator-option = :STRUCTURE:.

To specify a suffix, enter the following in your DB2 profile:

```
MPDY_INDICATOR_SUFFIX = :XX:
```

where *XX* represents any two alphanumeric characters. COBOL suffixes are generated as *-XX*. PL1 suffixes are generated as *_XX*. Assembler language suffixes are generated as *#XX*.

For example, to specify a suffix of *-DB* (in COBOL), *_DB* (in PL1), or *#DB* (in Assembler language), enter the following in your DB2 profile:

```
MPDY_INDICATOR_SUFFIX = :DB:
```

To stop any suffix being generated, enter the following in your DB2 profile:

```
MPDY_INDICATOR_SUFFIX = ::
```

Refer to the DB2 PRODUCE command in "DB2 PRODUCE" on page 273 for details of generating indicator structures and host language data structures.

Setting Suffixes Applied to Indicator Array Names

The variables MPDY_IND_ARRAY_SUFF_1 and MPDY_IND_ARRAY_SUFF_2 can now be set to user-defined suffixes for indicator arrays (previously possible for indicator structures only). For COBOL these suffixes correspond to levels 1 and 10 of the array. For PL/1 they correspond to levels 1 and 5 of the array. The defaults are:

```
MPDY_IND_ARRAY_SUFF_1 = :IS:
MPDY_IND_ARRAY_SUFF_2 = :IN:
```

If these variables are not set in the DB2 profile, or the profile is not executed, the same default values will apply.

Setting Suffixes Applied to Variable-Length Column Names

The variables MPDY_COB_SUFF_I and MPDY_COB_SUFF_2 can now be set to user-defined suffixes for the name generated when a column has a variable-length definition. For COBOL these suffixes correspond to the first and second instance of level 49 in the host structure. The defaults are:

```
MPDY_COB_SUFF_1 = :L:
MPDY_COB_SUFF_2 = :D:
```

If these variables are not set in the DB2 profile, or the profile is not executed, the same default values will apply.

Automatically Generating SQL COMMENT ON/LABEL ON Statements

You can tailor the output so that whenever an SQL CREATE statement is generated, SQL COMMENT ON/LABEL ON statements are also generated automatically from the same member, increasing processing speed.

To automatically generate SQL COMMENT ON statements, enter the following in your DB2 profile:

```
MPDY_CM_COMMENTOPT = :ON:
```

To automatically generate SQL LABEL ON statements, enter the following in your DB2 profile:

```
MPDY_CM_LABELOPT = :ON:
```

To generate SQL COMMENT ON or LABEL ON statements by themselves, use either the DB2 COMMENT and DB2 LABEL commands or the relevant export panel. To return to the default, in which SQL CREATE, COMMENT ON, and LABEL ON statements are generated separately, either remove the relevant directives or alter them to read:

```
MPDY_CM_COMMENTOPT = :OFF:
MPDY_CM_LABELOPT = :OFF:
```

Note:

You can use the WITH-LABELS and WITH-COMMENTS keywords in the DB2 CREATE command to similar effect for individual commands (overriding any setting in the DB2 profile).

Refer to "DB2 COMMENT and DB2 LABEL" on page 206 for details of the DB2 COMMENT and DB2 LABEL commands, and "DB2 CREATE" on page 212 for details of the CREATE command.

Generating One-, Two-, or Three-part Names for DB2 Objects

You can tailor output to produce one-, two-, or three-part names for DB2 objects when generating output.

To produce one-part names for DB2 tables, views, aliases, indexes, and packages, enter the following in your DB2 profile:

```
MPDY_CM_NAME_QUAL = 1
```

Note:

This does not apply to SQL GRANT and REVOKE statements generated.

To produce two-part names (the default), either remove the directive, or enter the following in your DB2 profile:

```
MPDY_CM_NAME_QUAL = 2
```

To produce three-part names (that is, names including a location) for DB2 tables and views, enter the following in your DB2 profile:

```
MPDY_CM_NAME_QUAL = 3
```

Note:

You can usually override these settings for individual DB2 export functions, using the SQLID and/or LOCATION keywords. However, if you set MPDY_CM_NAME_QUAL to 1, this is not true for these member types:

Keywords Specified	Member Type performed on	Result generated
LOCATION keyword only	DB2-TABLE DB2-VIEW	1-part name only.
LOCATION keyword only	DB2-INDEX	1-part name for the index, 3-part name for the table
SQLID and LOCATION keywords	DB2-INDEX	2-part name for the index, 3-part name for the table.

Refer to Chapter 8, "Commands," on page 175 for details of export to DB2 commands.

Setting a Tolerance Level for Output

You can set the level of tolerance for writing output. Output with messages of a severity below this level is accepted and written. Output with messages at or above this level is not written.

Information (I) messages have a severity level of 0, Warning (W) messages 4, and Error (E) messages 8. The default tolerance level is 8, which accepts I and W messages, but E messages cause output not to be written.

To set the tolerance level, enter the following in your DB2 profile:

```
MPDY_CM_SENDF_THRESH = level
```

where *level* is the tolerance level.

Setting the SQL Escape Character

To alter the SQL escape character, enter the following in your DB2 profile:

```
MPDY_CM_SQL_ESCAPE = char
```

where *char* is the escape character you want. The default is a single quote (').

Setting Width of Output for SQL COMMENT ON Statements

To set the standard left and right margins for SQL COMMENT ON output, enter the following in your DB2 profile:

```
MPDY_CM_COMMENT_LM = left-margin
```

and

```
MPDY_CM_COMMENT_RM = right-margin
```

where:

left-margin is the value of the left margin. The default is 1.

right-margin is the value of the right margin. The default is 72.

Setting Width/Indent of the SQL DROP Impact Analysis Report

To set the standard width of this report, enter the following in your DB2 profile:

```
MPDY_CM_DR_WIDTH = width
```

where *width* is the value of the width. The default is 72.

To set the standard indent of this report, enter the following in your DB2 profile:

```
MPDY_CM_DR_INDENT = indent
```

where *indent* is the value of the indent. The default is 5.

Allowing an Optional Space Character when Generating SQL DECIMAL Datatypes

You can tailor output to set generated SQL DECIMAL(*n,m*) datatypes to have a space character after the comma. In other words, the datatypes could be of type SQL DECIMAL(*n, m*).

To allow a space character, enter the following directive in your DB2 profile:

```
MPDY_CM_DCML_SPACE = 1
```

To return to the default, which does not allow a space character, either remove the directive, or replace it by the following:

```
MPDY_CM_DCML_SPACE = 0
```

Accessing a Specific DB2 Subsystem or Plan

When you initially establish access to a DB2 environment in a session, you can choose to access a specified DB2 subsystem, and to use a specified plan. If you do not specify a DB2 subsystem or plan, then the defaults for these are used. When you connect to a subsystem, you cannot then re-connect to a different subsystem during the same session: you must log off first.

To connect to a specific DB2 subsystem, enter the following in your DB2 profile:

```
MPDY_DB2SSN = :subsystem-name:
```

where *subsystem-name* is the name of the subsystem to connect to.

To specify a (non-default) plan for your current session, enter the following in your DB2 profile:

```
MPDY_PLANNAME = :plan-name:
```

where *plan-name* is the name of the plan.

Note:
MPDY_DB2SSN and MPDY_PLANNAME are both global variables.

Refer to your installation manual for further details of setting up access to a DB2 subsystem or plan.

Setting EXPORT Generated Object-name Length

You can tailor variables to stipulate the maximum length allowed for the object-name generated during the DB2 export.

To set the maximum length of the object name generated during DB2 PRODUCE COBOL FROM member-name:

```
MPDY_CM_COBLENGTH = :length:
```

The default for the above is 27.

To set the maximum length of the object name generated during DB2 PRODUCE ASSEMBLER FROM member-name:

```
MPDY_CM_BALLENGTH = :length:
```

The default for the above is 8.

To set the maximum length of the object name generated during DB2 PRODUCE PL1 FROM member-name:

```
MPDY_CM_PL1LENGTH = :length:
```

The default for the above is 28.

Setting the Generated Column Data Type

You can specify the data type of the column generated from an item with form-description NUMERIC-CHAR. Valid options are DEC or CHAR.

To generate SQL columns with data type DECIMAL, enter the following in your DB2 profile:

```
MPDY_NUM_GEN = :DEC:
```

To generate SQL columns with date type CHAR, enter the following in your DB2 profile:

```
MPDY_NUM_GEN = :CHAR:
```

The default is DEC.

Creating an INSERT Statement for Stored Procedures

The default values of these variables are used if no explicit attributes exist in a PROGRAM member type when a DB2 CREATE command program members for a DB2 Version 5.*x* system is executed.

Introduction to User Exits

User exits are set points in ASG-supplied software at which you can call user exit routines. *User exit routines* are either Corporate or User executive routines. The process of calling these routines at user exits is known as *taking user exits*.

You can use user exits to tailor output by attaching your own executive routines to Manager Products, using the DB2 profile.

You can take user exits:

- To retrieve additional data when accessing a repository member. You can take this type of user exit to alter the contents of the WBTA (an area of storage containing command variables), by setting variables in the DB2 profile
- Every time you perform a specified DB2 export function (such as generating a CREATE TABLE statement), by altering the DB2 profile to take a named exit for that function
- For one DB2 export function.

Refer to Chapter 8, "Commands," on page 175 for details of DB2 export commands.

Taking User Exits when Accessing a Repository Member

This facility has been replaced by that described in "Setting Specific User Exits to be Taken" on page 126, but, to ensure full MIR3 compatibility, it will still operate.

When a repository member is being accessed by an export to DB2 function (via the DACCESS command), you can cause your own user exit routine to retrieve extra data, using the DRETRIEVE command. For example, you can obtain user defined attributes held in a member.

To set a user exit to be taken whenever a member is being accessed, enter the following in your DB2 profile:

```
MPDY_CM_OBJEXIT = :obj-exit:
```

where *obj-exit* is the name of a user exit routine.

The user exit is normally taken once for each member accessed. You can also take a user exit for each column in the member. For example, if you access a DB2-TABLE, containing three ITEMs documenting columns, you can take one exit for the DB2-TABLE (as described above) and one exit for each ITEM.

To set a user exit to be taken for each GROUP or ITEM member representing a column of a DB2 table, view, or index, enter the following in your DB2 profile:

```
MPDY_CM_COLEXIT = :col-exit:
```

where *col-exit* is the name of a user exit routine.

For this second type of user exit, the array index *i* (for the *i*'th column) is also passed to the user exit routine called.

Note:

If a GROUP contained within a DB2-TABLE, DB2-VIEW, or DB2-INDEX generates more than one column, the column exit is taken for each column within that GROUP.

When WBTA, OBJ, or COL are specified as type, then the variant required is DB2-PRODUCE-TABLE and DB2-PRODUCE-VIEW. When OUT is the specified type, in these two commands, then the above output variants should be used.

Taking User Exits For Specified DB2 Export Functions

You can set user exits to be taken for specific export to DB2 functions, using the COMMAND member MPDY ADDXIT in your DB2 profile. You could modify output, for example, to suit your installation standards.

All export to DB2 functions send output to a specified destination (such as the screen, or a USER-MEMBER on the MP-AID) via two buffers:

- The output buffer that contains the generated output itself.
- The messages buffer that contains associated messages.

The output buffer has pointers to the messages buffer.

In a user exit routine, you can access and manipulate both of these buffers, and all command variables relevant to the specified function, before the contents of the buffers are sent to their destination.

You can examine all these data structures, using the DB2 DEBUG command.

Refer to "Writing User Exit Routines" on page 128 for details of how to write exit routines to take advantage of these data structures to tailor your output.

Setting Specific User Exits to be Taken

To set a user exit to be taken whenever a specific export to DB2 function is executed, enter the following in your DB2 profile:

`:MPDYADDXIT` *`type variant routine`*`:`

where *`type`* is the type of export: WBTA, OUT, OBJ, or COL.

If WBTA is specified, then the exit is taken when the table column and constraint details have all been retrieved from the members and loaded into the work bench arrays. The exit is taken before any output processing occurs. There are no parameters passed.

If OBJ is specified, then the exit is taken when only the table details have been loaded, before any column details are available to be tailored. The exit is taken once for each member referred to via the AS clause. The parameter list is: MEMBERNAME.

If COL is specified, then the exit is called once as each column is Daccessed. The parameter list is: COLUMN NUMBER DACCESS-RETURN-CODE.

If OUT is specified, then the exit is called each time a line is written. See "Writing User Exit Routines" on page 128 for further details. The parameter list is: EXIT POINT LINE NUMBER.

`Variant` specifies the type of export to DB2 function (for example, generation of SQL ALTER INDEX statements). You can have an exit for each variant.

`Routine` is the name of the user exit routine you wish to call.

MPDYADDXIT is an ASG-supplied COMMAND member. This connects up your user exit routine to the type of export to DB2 function specified.

All variants have a starting prefix of DB2-. The second and successive elements of a variant name specify the version of the export to DB2 function.

For example, the variant DB2-PRODVIE-COB-FOR-WS specifies that a user exit is to be taken whenever you use the DB2 PRODUCE command on a DB2-VIEW member, to generate a COBOL data structure for working-storage.

The names of the output variants are listed below.

```
DB2-ALTER-INDEX      DB2-ALTER-STOGROUP
DB2-ALTER-TABLE      DB2-ALTER-TBSPACE
DB2-BIND-PACKAGE     DB2-BIND-PLAN
DB2-COMMENT-ALIAS    DB2-COMMENT-TABLE
DB2-COMMENT-VIEW     DB2-CREATE-ALIAS
DB2-CREATE-DATABASE  DB2-CREATE-INDEX
DB2-CREATE-STOGROUP  DB2-CREATE-TABLE
DB2-CREATE-TBSPACE   DB2-CREATE-VIEW
DB2-DECLARE-TABLE    DB2-DECLARE-VIEW
DB2-DROP-ALIAS       DB2-DROP-DATABASE
DB2-DROP-INDEX       DB2-DROP-STOGROUP
DB2-DROP-TABLE       DB2-DROP-TBSPACE
DB2-DROP-VIEW        DB2-GRANT-REVOKE-PRIVILEGE
DB2-LABEL-ALIAS      DB2-LABEL-TABLE
DB2-LABEL-VIEW       DB2-RECALC-TABLE
DB2-RECALC-INDEX     DB2-SIZE-TABLE
DB2-SIZE-INDEX       DB2-SYNONYM
```

where *type* is specified as OUT

```
DB2-PRODTAB-BAL-FOR-SQL
DB2-PRODTAB-BAL-FOR-WSDB2-PRODTAB-COB-FOR-SQL
DB2-PRODTAB-COB-FOR-WSDB2-PRODTAB-PL1-FOR-SQL
DB2-PRODTAB-PL1-FOR-WSDB2-PRODTAB-TAB-FOR-SQL
DB2-PRODTAB-TAB-FOR-WSDB2-PRODUCE-TABLE
DB2-PRODVIE-BAL-FOR-SQL
DB2-PRODVIE-BAL-FOR-WS
DB2-PRODVIE-COB-FOR-SOL
DB2-PRODVIE-COB-FOR-WS
DB2-PRODVIE-PL1-FOR-SQL
DB2-PRODVIE-PL1-FQR-WS
DB2-PRODVIE-TAB-FOR-SOL
DB2-PRODVIE-TAB-FOR-WS
```

where *type* is specified as WBTA, OBJ, or COL

```
DB2-PRODUCE-TABLEDB2-PRODUCE-VIEW
```

Writing User Exit Routines

User exit routines for export functions take two input arguments:

- The current exit point (the logical point in output processing from which user exit is called)
- The current output line number; this is supplied for exit points 2 and 3.

User exits are taken at these exit points during output:

1 Before any lines are written, with the exit_point parameter set to 1.

2 Before line 1 is written, with exit_point set to 2. The line number is also passed to the user exit as a parameter.

3 After line 1 is written, with exit_point set to 3. The line number is also passed to the user exit as a parameter.

The user exit is then re-taken, repeating steps 2 and 3 (with exit_point set to 2 then 3 respectively) for each of the remaining lines.

▸ After all output lines are written.

You can access all command variables in your user exit routine.

These variables will be important to you:

Variable	Description
MPDY_MAX_ERROR	Set to 0, 4, 8, or 12 (for **I**nformation, **W**arning, **E**rror or **S**evere messages). This shows the highest error level in the output buffer (or the highest level so far, depending on when it is used during execution).
MPDY_OUT_LINE (*x*)	The contents of the current line (*x* is the current line number, supplied as a parameter).

Variable	Description
MPDY_OUT_KEY (*x*)	Specifies whether to print or suppress the current line and associated messages. Can be set to 0, 1, 2, or 3: 0: suppress the current line and associated messages 1: print output line, but suppress messages 2: print messages, but suppress output line 3: print both output line and messages (the default)
MPDY_OUT_ SOURCE	Shows the type of information held in the current line. This is usually a prefix.
MPDY_OUT_SOURCE_OCC	The occurrence within the data structure described by MPDY_OUT_SOURCE from which the output line was generated.

You can examine these and other variables using the DB2 DEBUG command.

ASG supplies the example user exit routines MPDYl2OXCR, MPDY12OXPB, MPDYl2OXPC, MPDYl2OXPP, and MPDYI2OXCV. We recommend that you examine these routines before writing your own.

Taking User Exits for an Individual Export Function

You can take a user exit for an individual export to DB2 function.

For example, to take a user exit when generating an SQL CREATE TABLE statement for the DB2-TABLE member TB-DJB-CUST, calling the user exit routine MPDY12OXCR, enter:

```
DB2 CREATE TB-DJB-CUST USING MPDY120XCR ;
```

This method of taking user exits gives you more flexibility, although you have to call each user exit routine each time you wish to use it.

These user exit routines are handled similarly to those set up by altering the DB2 profile using the COMMAND member MPDYADDXIT. You should therefore write them in a similar manner.

Refer to "Writing User Exit Routines" on page 128 for details of writing user exit routines. Refer to Chapter 8, "Commands," on page 175 for details of export to DB2 commands.

Setting Switches for Non-standard SQL Generation

Switches exist to generate non-standard SQL statements (statements generated from a previous Manager Products release) back to the previous (perhaps erroneous) syntax. This allows you to generate SQL statements (even if these statements are not fully compliant with the SQL standard):

- MPDY_TAB_WITH_DEFAULT_IS_NOT_NULL.
  - Is set to Y to cause any column that has the clause WITH-DEFAULT to automatically generate WITH DEFAULT NOT-NULL in the SQL statement for that column.
  - Is set to N to only generate WITH DEFAULT. The SQL clause NOT NULL is only generated if the column definition explicitly states NOT-NULL in the member definition.

  The value N is the assumed default setting in accordance to the SQL standard.
- MPDY_PRIMARY_KEY_IS_NOT_NULL
  - Is set to Y to force automatic addition of NOT NULL to all columns that are primary-key-columns in the definition of a DB2 table.
  - Is set to N to ensure that NOT NULL is only generated when the member definition does contain the clause NOT-NULL explicitly.

  The value N is the assumed default setting in accordance with the SQL standard.

Note:

The use of these switches may generate incorrect SQL syntax or make correct syntax invalid if applied to specific relational DB systems.

6 Dynamic Import/Export

This chapter includes these sections:

Security and Authorization 133

Output 133

Using Executive Routines with Dynamic SQL Functions 134
Variables Used for Dynamic Import/Export 139
Control Variables 139
Return Variables 140
COMMAND and EXECUTIVE Members Used in Dynamic SQL Functions . 141
Creating and Populating a Table 141
Inserting Rows into a Table 142
Importing Information and Assigning it to Command Variables 144
Submitting Any SQL Statement that Can Be Prepared 146
Creating Your Own HELP Text 147

You can submit SQL statements to your DB2 or SQL/DS environment and receive the results, from within Manager Products, using dynamic SQL functions. These functions are provided by the ISQL command or in executive routines.

To use dynamic SQL functions, you must have Manager Products and either DB2 or SQL/DS available on the same CPU and operating system.

You can select a DB2 subsystem or plan to use when initially establishing access to your DB2 environment.

Refer to "Setting Suffixes Applied to Indicator Array Names" on page 119 for details of accessing a specific DB2 subsystem or plan.

You can submit any SQL statement which can be dynamically prepared for execution. SQL SELECT statements must conform to the specifications of a full select statement.

For example, you can submit SQL statements generated by a previous Manager Products DB2 or SQL command by using the ISQL command.

Refer to Chapter 8, "Commands," on page 175 for details of the ISQL and DB2 commands.

You can also create executive routines using the Manager Products Procedures Language to:

- Update your DB2 or SQL/DS environment with information held in the repository
- Import information from DB2 or SQL/DS into the repository

from within your Manager Products environment. Refer to "Using Executive Routines with Dynamic SQL Functions" on page 134 for details of how to create these executive routines.

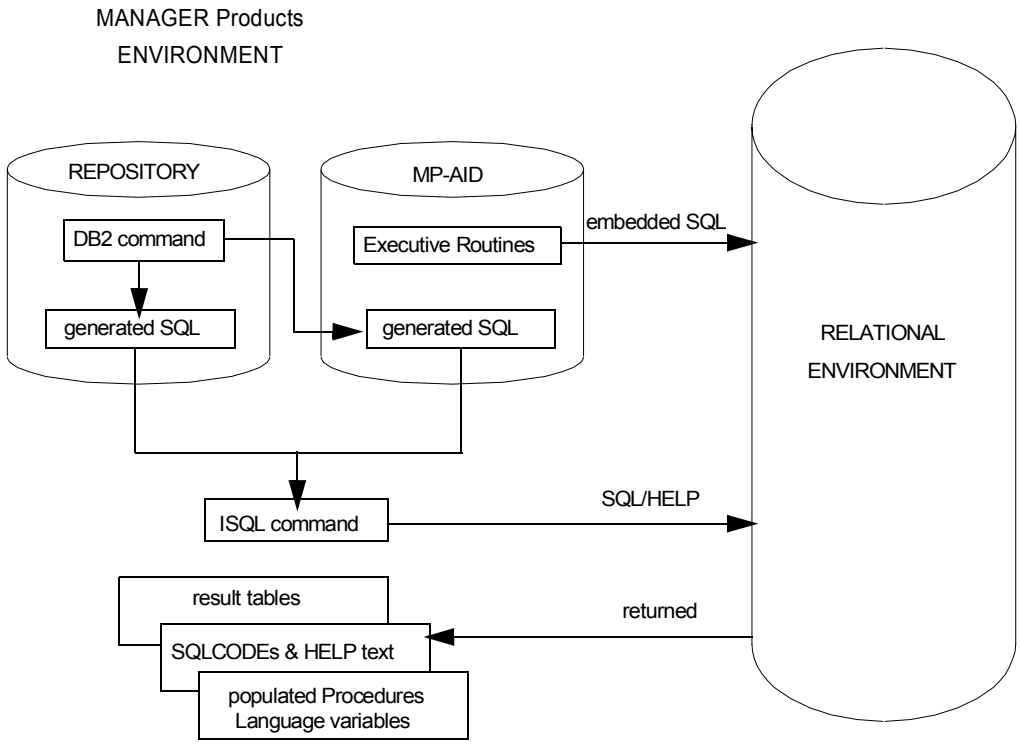
Refer to the *ASG-Manager Products Procedures Language* manual for details of the Procedures Language.

SQL statements can be submitted both interactively and in batch.

You therefore have a direct link between your relational environment and your Manager Products environment.

Figure 25 illustrates the dynamic SQL functions.

Figure 25 • Dynamic SQL functions



Security and Authorization

The SQL statements you can submit using dynamic SQL functions are determined by the privileges granted to your authorization ID in DB2 or SQL/DS. Your authorization ID is your operating system logon in the environment in which DB2 or SQL/DS operates.

For example, if SQL/DS is operating in a VM environment, the authorization ID is the CMS logon. For DB2 environments operating in a TSO environment, it is the TSO logon.

Output

Output varies depending on your relational environment, the method by which you are submitting SQL statements, and whether or not the SQL statements are successful.

SQL statements submitted with an ISQL command are printed. SQL statements submitted from within executive routines are not printed.

A result table is printed in response to a successful SQL SELECT statement submitted with the ISQL command or from within an executive routine calling the COMMAND member MPDYDSSSQL.

Result tables are not printed in response to successful SELECT statements submitted from within an executive routine calling the COMMAND member MPDYDSSSEL.

You can limit the number of rows to be included in a result table by specifying an integer in the ISQL command or by including the variable SQLLROWS in an executive routine.

If you do not specify a maximum number of rows then the number printed is determined by the maximum number of lines of output that can be printed in any output buffer. The maximum line limit does not affect result tables printed in response to SQL statements submitted in batch. Use the QUERY OUTPUT-LINE-LIMIT command to find the maximum line limit.

Refer to the *ASG-ControlManager User's Guide* for details of the QUERY OUTPUT-LINE-LIMIT command.

If a row in a result table cannot be printed on a single line on the current output device then the row will wrap around to the next line. Each column in a result table is truncated to a width of thirty characters unless the result table contains two or less columns in which case the full width of columns is displayed.

A question mark (?) in a result table indicates that a value in a column is null or that the value is of a data type that cannot be printed within the Manager Products environment. You can display values with non-printable data types by specifying an SQL Scalar Function in the SELECT statement in order to change the representation of the value.

To display values with data types of TIME, TIMESTAMP, or DATE in a result table, you must include a CHAR function in the SELECT statement. To display values with data types of FLOAT in a result table you must include a DECIMAL function in the SELECT statement.

Commas are, where appropriate, included in values having an INTEGER data type when the value is displayed in a result table.

You can create an executive routine which generates result tables in a format which suits your environment by calling the COMMAND member MPDYDSSSQL and using the Procedures Language to tailor the output it returns.

A DB2 or SQL/DS SQLCODE is displayed in response to any unsuccessful SQL statements you have submitted. SQL/DS SQLCODEs are followed by explanatory SQL/DS HELP text. DB2 SQLCODES are not followed by HELP text.

The systems administrator can create HELP text for both DB2 and SQL/DS which suits your own environment by tailoring the EXECUTIVE member MPDYDSSXIT. Tailored HELP text is only displayed in response to SQL statements submitted from within an executive routine.

The systems administrator can also specify the DB2 subsystem to be connected to when initially accessing DB2, and the plan to use.

Refer to "COMMAND and EXECUTIVE Members Used in Dynamic SQL Functions" on page 141 for details of the COMMAND and EXECUTIVE members provided by dynamic SQL functions.

Using Executive Routines with Dynamic SQL Functions

You can create executive routines which dynamically submit embedded SQL statements to your DB2 or SQL/DS environment.

By combining SQL statements and the Manager Products Procedures Language you can create executive routines which can:

- Create and populate a new table
- Insert rows into an existing table
- Import information from tables and views into the Manager Products environment
- Submit any SQL statement that can be dynamically prepared for execution

The different executive routines carrying out these tasks must call particular COMMAND and EXECUTIVE members and contain particular Procedures Language command variables.

The command variables contain the information transferred between Manager Products and your relational environment by the executive routine.

When creating and populating a new table, or inserting rows into an existing table, the command variables define the objects to be created. For example, a variable could define a column and the values assigned to the different elements of the variable would define the values in the column. You can use the DACCESS and DRETRIEVE commands to assign information filed in the repository to the variables.

When importing information into the Manager Products environment the information about a particular object is assigned to the relevant variable. You can use the Procedures Language to manipulate the imported information. For example, you can generate result tables and display them in your own format.

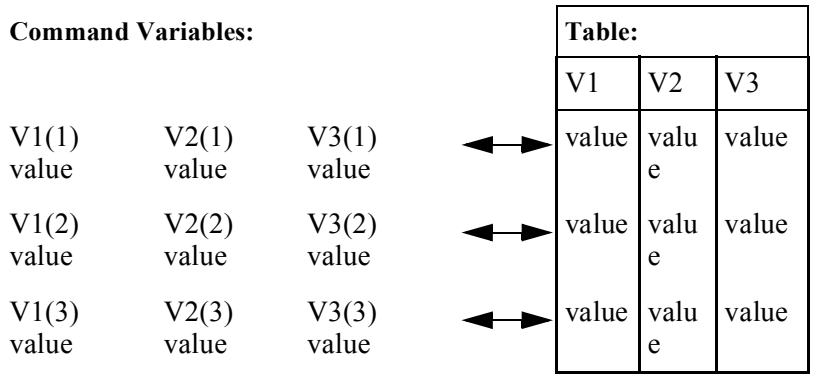
By tailoring the EXECUTIVE member MPDYDSSXIT the systems administrator can create HELP text which is printed in response to the SQL statements you have submitted from within an executive routine.

Refer to the *ASG-Manager Products Procedures Language* manual for details of the DACCESS and DRETRIEVE commands.

Variables and the Column and Row Structure of Tables and Views

Each column in a table or view is represented within an executive routine by a command variable. command variables are arrays, with a maximum of 60,000 separate array elements. Each element represents a value in the column.

For example, three command variables named VI, V2, and V3, each having three elements, could represent a table with three columns and rows as follows:



If the command variables in an executive routine used to create or insert rows into a table each have different numbers of elements, then the number of rows in the table will equal the array with the maximum number of elements, and null values are entered in those columns for which no element was specified.

For example, the command variables V1, V2, and V3 (V1 having three elements, V2 two elements, and V3 one element) could be used to create a table with three columns and rows as follows:

Command Variables:

V1(1) value	V2(1) value	V3(1) value
V1(2) value	V2(2) value	
V1(3) value		

Table:

V1	V2	V3
value	value	value
value	value	null
value	null	null

You can import result tables from SQL SELECT statements. The values in columns are assigned to command variables which you can name. As many elements are created for these variables as are required to contain all the values in a column. Null values are assigned to command variables as undefined (null) values.

For example, information from a table with null values would be assigned to the elements of the command variables V1, V2, and V3 as follows:

Command Variables:

V1(1) value	V2(1) value	V3(1) value
V1(2) value	V2(2) null	V3(2) value
V1(3) value	V2(3) value	V3(3) null

Table:

V1	V2	V3
value	value	value
value	null	value
value	value	null

How to Define the Data Type and Values of Columns

You can create executive routines which create and populate, or insert rows into, a table.

The columns in a table are defined within the executive routine by command variables. The values within the column are defined by the values assigned to the different elements of the Procedures Language command variable. The data type of the column is determined by the value assigned to the first element of the command variable.

Command variables can only be assigned a numeric or character value. Character values include alphanumeric strings. A column is defined as having an INTEGER data type if the first element of the variable is assigned a numeric value or if it is not assigned any value. A column is defined as having a VARCHAR 254 data type if the first element of the variable is assigned a character value.

For example, you could define two columns and the values they contain by including the following variables and directives in an executive routine:

```
COMMAND AREA
COMMAND QTY
AREA(1) = NORTH
AREA(2) = SOUTH
QTY(2) = 550
```

The first column would be called AREA and have a data type of VARCHAR(254). The column would contain the two character values NORTH and SOUTH.

The second column would be called QTY and have a data type of INTEGER. The column would contain two values, the first of which is null and second of which has a numeric value of 550.

If the first value in a column is null it can only be followed by numeric values. The column is defined as having a data type of INTEGER.

Character values cannot be entered in a column which has been defined as having a data type of INTEGER. The character value and any subsequent values are not entered in the table.

A numeric value entered in a column which has been defined as having a data type of VARCHAR(254) is treated as a character value and is prefixed with zeros. Subsequent values are entered in the table.

Command variables can be assigned character values that are a maximum of 255 characters long. Character values are, if necessary, truncated to 254 characters when entered in a column having a data type of VARCHAR(254).

To create or insert rows into tables with columns having data types other than INTEGER or VARCHAR(254) you can either use the ISQL command with appropriate SQL commands (for example, CREATE TABLE), or create an executive routine calling the COMMAND member MPDYDSSSQL.

Importing Information from Columns with Particular Data Types

You can create an executive routine to import information from tables and views into the Manager Products environment. The values in each column are assigned to command variables as Procedures Language values. Command variables can be assigned numeric or character values. Numeric values can have a maximum value of 2,147,483,647 and a minimum value of -2,147,483,648. Character values can be a maximum of 255 characters long.

Only information that can be assigned to a variable as a character or numeric value can be imported. For example, the values in columns with data types of GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC cannot be imported.

Information cannot be imported from columns with a data type of TIME, TIMESTAMP, or DATE unless you use the SQL CHAR function to obtain a character representation of the value.

Information cannot be imported from columns with a data type of FLOAT unless you use the SQL DECIMAL function to obtain a numeric representation of the value.

Values in columns with a data type of VARCHAR or LONG VARCHAR are, if necessary, truncated to 255 characters.

Values in columns with a data type of FLOAT or DECIMAL are, if necessary, truncated after the decimal point:

Table 18 Column Data Types and Procedures Language Values

Column Data Type	Procedures Language Value
TIME	CHARACTER (using the CHAR function)
TIMESTAMP	CHARACTER (using the CHAR function)
DATE	CHARACTER (using the CHAR function)
GRAPHIC(*n*)	Not supported
VARGRAPHIC(*n*)	Not supported
LONG VARGRAPHIC(*n*)	Not supported
CHAR(*n*)	CHARACTER
VARCHAR(*n*)	CHARACTER
LONG VARCHAR	CHARACTER
SMALLINT	NUMERIC
INTEGER	NUMERIC
FLOAT(*n*)	NUMERIC (using the DECIMAL function)

Table 18 Column Data Types and Procedures Language Values

Column Data Type	Procedures Language Value
DECIMAL(*n*,*m*)	NUMERIC
BLOB/CLOB/DBCLOB (*n*)	CHARACTER
ROWID	CHARACTER USAGE ROWID

The above table shows the type of value which the information in columns is given when assigned to Manager Products Procedures Language command variables.

Variables Used for Dynamic Import/Export

Command variables contain the information that is transferred between Manager Products and your relational environment by the executive routines you have created. The variables can be divided into *control variables* and *return variables*.

Control variables. Command variables which are specified in executive routines. For example, the SQLI_ COMMAND variable specifies the SQL statements to be submitted.

Return variables. Command variables which are not specified in executive routines but are generated by dynamic SQL functions in response to the SQL statements you have submitted. For example, the SQLI_ CODE(1) variable which contains the SQLCODE number returned from your relational environment.

Control Variables

SQLI_COMMAND defines the SQL statement you want to submit. You must specify additional elements of the variable if you need to continue the statement. Statements defined in the additional elements must start with a space otherwise the whole statement will concatenate and be rejected by DB2 or SQL/DS.

If you repeat the SQLI_COMMAND variable in an executive routine you must ensure that you have no unwanted data still set from a previous use of the command variable. For example, if the first SQL statement you submit uses three elements of the SQLI_COMMAND variable but the second SQL statement only uses two elements, you must set the third element to null. The easiest way to do this is to drop SQLI_COMMAND and re-declare it before use.

SQLI_TABLE_SPACE defines the name of the DB2 table space or SQL/DS dbspace in which a table you want to create will be stored. The dbspace or table space must already exist.

SQLI_TABLE_NAME defines the name of a table. The table name can be qualified or unqualified. DB2 or SQL/DS adds an implicit qualifier if it is unqualified. The implicit qualifier will be your logon in the environment in which DB2 or SQL/DS operates. If you are creating and populating a new table then a table of the same name must not already exist. If you are inserting rows in an existing table then the table must exist.

SQLI_ROWS defines the maximum number of rows to which the executive routine will be applied. For example, the number of.rows to be displayed in a result table or the number of rows to be populated or inserted into a table. The limit overrides the number of rows in the result table or the number of elements in the command variables defining the rows to be populated or inserted.

When creating tables or inserting rows into a table you must specify command variables naming the columns in the table. When importing information from tables and views you can name command variables to which the imported information will be assigned.

Return Variables

SQLI_CD_*n* (*n* is the number or the column) contains the values in the columns from which you have imported information. This is generated as a default if you have not named the variables to which the information is to be assigned.

SQLI_CS contains the maximum size of the columns from which you have imported information. The size is calculated in bytes. The maximum size is either the column name or the largest value in the column, whichever is greater.

SQLI_CL contains the names of the columns from which you have imported information.

SQLI_RETURN contains the Manager Products code returned by dynamic SQL functions. The return code is the same as the number of the Manager Products message it generates.

SQLI_SQLCODE(1) contains the SQL/DS or DB2 SQLCODE returned from your relational environment via the SQL Communication Area (SQLCA).

SQLI_SQLCODE(2) contains the SQL/DS or DB2 SQLERRM returned from your relational environment via the SQL Communication Area (SQLCA).

The SQLI_RETURN, SQLI_CODE(1), and SQLI_CODE(2) variables are examined by the EXECUTIVE member MPDYDSSXIT which you can tailor to generate HELP text to suit your environment, or perform additional checking.

COMMAND and EXECUTIVE Members Used in Dynamic SQL Functions

These COMMAND and EXECUTIVE members must be called from the executive routines with which you submit embedded SQL statements to your DB2 or SQL/DS environment.

- MPDYDSSCRT: creating and populating a table
- MPDYDSSINS: inserting rows into a table
- MPDYDSSSEL: importing information and assigning it to Procedures Language variables
- MPDYDSSSQL: submitting any SQL statement that can be dynamically prepared for execution
- MPDYDSSXIT: displaying SQLCODEs and SQLERRMs.

Creating and Populating a Table

You can create an executive routine which creates a table and specifies the DB2 table space or SQL/DS dbspace it is stored in, the names of its columns, the number of rows it contains, and the values within each column.

The executive routine must call the COMMAND member MPDYDSSCRT to create and populate the table, the EXECUTIVE member MPDYDSSXIT to display any SQLCODEs and HELP text returned from your relational environment, and must contain the command variables SQLI_TABLE_ NAME, SQLI_TABLE_ SPACE, and those variables defining the columns in the table.

The executive routine can also optionally contain the command variable SQLI_ROWS, and those directives defining the values in the columns.

The called COMMAND member MPDYDSSCRT must be followed in the executive routine by the names of the columns you have defined. The call must be enclosed in literal delimiters.

By specifying DACCESS and DRETRIEVE commands in the executive routine you can populate the table with information filed in members in the repository.

For example, to create a table named SALES stored in a table space or dbspace named COMPANY and containing the following columns and rows:

AREA	CATALOGUE_NAME	QTY
NORTH	D35	null
SOUTH	D36	5500

enter the executive routine in Figure 26:

Figure 26 • Example Executive Routine to Create and Populate a Table

```
MPXX LITERAL=:
/*_____________________________________________________________________
/*  Example Executive Routine to create & populate a table.
/*_____________________________________________________________________
/*  Naming the table and dbspace or table space.
/*_____________________________________________________________________
COMMAND SQLI_TABLE_NAME
COMMAND SQLI_TABLE_SPACE
SQLI_TABLE_NAME  = :SALES:
SQLI_TABLE_SPACE = :COMPANY:
/*_____________________________________________________________________
/* Specifying the number of rows to be created in the table
/*_____________________________________________________________________
COMMAND SQLI_ROWS
SQLI_ROWS = 2
/*_____________________________________________________________________
/* Naming the columns.
/*_____________________________________________________________________
COMMAND CATALOGUE_NAME
COMMAND QTY
COMMAND AREA
/*_____________________________________________________________________
/* Populating the columns.
/*_____________________________________________________________________
DACCESS MEMBER :GROUP-SALES-NS:;
DRETRIEVE ALL ALIAS;
DRETRIEVE ALL CATALOGUE;
QTY(2) = :5500:
AREA( ) = ALIAS_NAME
/*_____________________________________________________________________
/* Call COMMAND member MPDYDSSCRT, EXECUTIVE member MPDYDSSXIT.
/*_____________________________________________________________
MPDYDSSCRT  :AREA: :CATALOGUE_NAME: :QTY: ;
MPDYDSSXIT;
/*_____________________________________________________________________
/* Either exit or create and populate another table.
/*_____________________________________________________________________
EXIT
```

Inserting Rows into a Table

You can create an executive routine which inserts rows into a table. This executive routine must call:

- The COMMAND member MPDYDSSINS to insert rows into the table, followed in the executive routine by the names of the columns in the table into which you are inserting rows. Enclose the call in literal delimiters.
- The EXECUTIVE member MPDYDSSXIT to display any SQLCODEs and HELP text returned from your relational environment,

and must contain the command variables SQL_TABLE_NAME, SQLI_TABLE_SPACE, and those variables and directives giving the names of the columns and defining the rows to be inserted. Also, the executive routine can optionally contain the command variable SQLI_ROWS.

By specifying DACCESS and DRETRIEVE commands in the executive routine you can insert information filed in members in the repository into the table.

For example, enter the executive routine in Figure 27 to insert the third and fourth rows into the table below (SALES):

AREA	CATALOGUE_NAME	QTY
NORTH	D35	null
SOUTH	D36	5500
EAST	null	null
WEST	D37	null

Figure 27 • Example Executive Routine to Insert Rows in a Table

```
MPXX LITERAL=:
/*  Example Executive Routine to insert rows into a table.
/*  Naming the table and table space or dbspace.
/*
COMMAND SQLI_TABLE_NAME
COMMAND SQLI_TABLE_SPACE
SQLI_TABLE_NAME  = :SALES:
SQLI_TABLE_SPACE = :COMPANY:
/*_____________________________________________________________________
/* Specifying the number of rows to be inserted.
/*_____________________________________________________________________
COMMAND SQLI_ROWS
SQLI_ROWS  =  2
/*_____________________________________________________________________
/* Specifying the column names.
/*_____________________________________________________________________
COMMAND AREA
COMMAND CATALOGUE_NAME
COMMAND QTY
/*_____________________________________________________________________
/* Defining the rows to be inserted.
/*_____________________________________________________________________
DACCESS MEMBER :GROUP-SALES-EW:;
DRETRIEVE ALL ALIAS;
AREA( )  =  ALIAS_ NAME
CATALOGUE_NAME(2)  =  D37
/* Call COMMAND member MPDYDSSINS and EXECUTIVE member MPDYDSSXIT.
MPDYDSSINS  :AREA: :CATALOGUE_NAME: :QTY: ;
MPDYDSSXIT;
/* Either exit or insert rows into another table.
EXIT
```

Importing Information and Assigning it to Command Variables

You can create an executive routine which imports information from tables and views and assigns it to Procedures Language command variables.

The executive routine must call:

- The COMMAND member MPDYDSSSEL to import the information
- The EXECUTIVE member MPDYDSSXIT to display any SQLCODEs and HELP text returned from your relational environment

and contain the command variable:

- SQLI_COMMAND

and can optionally contain:

- The command variable SQLl_ROWS
- The names of the command variables to which the imported column values will be assigned.

If you do not name the variables to which column values are to be assigned, they are given the default name SQLI_CD_*n* where *n* is the number of the column.

The called COMMAND member MPDYDSSSEL must be followed in the executive routine by the names of the command variables you have named. The call must be enclosed in literal delimiters. You do not need to specify the default names if you have not named any variables.

The executive routine submits your SQL SELECT statements to your relational environment. The result of the SELECT statement is used to populate command variables. Because you can submit any SELECT statement which conforms to the specifications of a full select statement, you can import information from several tables, views, and expressions. For example, you can include in the SELECT statement, joins from several tables, and views, scalar functions and other operators such as UNIONs and ORDER BY.

You can manipulate the information assigned to the command variables by using the Manager Products Procedures Language. For example, you can generate result tables in a format which suits your environment.

For example, to import information about a table named SALES and then display the information in the following format:

```
AREA CATALOGUE_NAME QTY
NORTH D35
SOUTH D36           5500
```

enter the executive routine in Figure 28:

Figure 28 • Example Executive Routine for Importing Information and Assigning it to Command Variables

```
MPXX LITERAL=:
/*_____________________________________________________________________
/*  Example Executive Routine for importing information and assigning it
/*  to Command Variables.
/*_____________________________________________________________________
/*  Naming the variables to which the information is to be assigned.
/*_____________________________________________________________________
COMMAND AREA
COMMAND CATALOGUE_NAME
COMMAND QTY
/*_____________________________________________________________________
/* Specifying the number of rows of information to be imported.
/*_____________________________________________________________________
COMMAND SQLI_ROWS
SQLI_ROWS  =  2
/*_____________________________________________________________________
/* Specifying the SELECT statement.
/*_____________________________________________________________________
DROP SQLI_COMMAND
COMMAND SQLI_COMMAND
SQLI_COMMAND(1)  =  :SELECT AREA, CATALOGUE_NAME, QTY:
SQLI_COMMAND(2)  =  :FROM SALES:
/*_____________________________________________________________________
/* Call COMMAND member MPDYDSSSEL.
/*_____________________________________________________________________
MPDYDSSSEL :AREA: :CATALOGUE_NAME: :QTY: ;
/*_____________________________________________________________________
/* Call EXECUTIVE member MPDYDSSXIT and exit.
/*_____________________________________________________________________
MPDYDSSXIT ;
IF SQLI_RETURN  =  0 THEN GOTO NOERRORS
EXIT
-NOERRORS
/*_____________________________________________________________________
/* Display the result table according to our specs:
/*_____________________________________________________________________
LOCAL 1
DO AREA( )
  I  =  FDO(:DARRAY:)
  SAY LEFT(AREA(I),SQLI_CS(1)) -
  LEFT(CATALOGUE_NAME(I),SQLI_CS(2)) -
  LEFT (QTY(I),SQLI_CS(3))
END
EXIT
```

Submitting Any SQL Statement that Can Be Prepared

You can create an executive routine which submits to your relational environment any SQL statement that can be dynamically prepared for execution.

The executive routine must call:

- The COMMAND member MPDYDSSSQL to submit the SQL statement
- The EXECUTIVE member MPDYDSSXIT to display any SQLCODEs and HELP text returned from your relational environment

and contain the command variable:

- SQLI_COMMAND

and can optionally contain the command variable:

- SQLI_ROWS.

Because you do not have to define columns with command variables you can submit SQL statements to tables having columns of any data type. For example, you can insert rows into tables having columns with data types other than INTEGER or VARCHAR(254).

For example, to add a column named DELIVERY with a data type of DATE to a table named SALES, enter the executive routine in Figure 29:

Figure 29 • Example Executive Routine to Generate any SQL Statement That Can be Prepared

```
MPXX LITERAL=:
/*_____________________________________________________________________
/* Example Executive Routine to generate any SQL statement that can be prepared.
/*_____________________________________________________________________
/*  Specifying the SQL statement.
/*_____________________________________________________________________
DROP SQLI_COMMAND
COMMAND SQLI_COMMAND
SQLI_COMMAND(1)  =  :ALTER TABLE SALES:
SQLI_COMMAND(2)  =  :  ADD DELIVERY DATE:
/*_____________________________________________________________________
/* Call COMMAND member MPDYDSSSQL.
/*_____________________________________________________________________
MPDYDSSSQL ;
/*_____________________________________________________________________
/* Call EXECUTIVE member MPDYDSSXIT.
/*_____________________________________________________________________
MPDYDSSXIT ;
/*_____________________________________________________________________
/* Either exit or submit another SQL statement.
/*_____________________________________________________________________
EXIT
```

Creating Your Own HELP Text

DB2 or SQL/DS SQLCODEs are printed in response to any unsuccessful SQL statements you have submitted to your relational environment. SQL/DS SQLCODEs are followed by explanatory HELP text. DB2 SQLCODEs are not followed by explanatory HELP text.

By tailoring the EXECUTIVE member MPDYDSSXIT you can specify the HELP text to be displayed in response to each SQLCODE number and so create your own HELP text to suit your own environment.

For example, if you were to include the following Procedures Language directive in the member MPDYDSSXIT:

```
IF SQLI_CODE(1) = -204 THEN SAY SQL_CODE(2)  :NOT FOUND:
```

then the HELP text, *variable-name* `NOT FOUND`, would be displayed in response to the SQLCODE number -204.

In the above example, *variable-name* is a name DB2 or SQL/DS has returned with the SQLCODE -204.

You can also make the member MPDYDSSXIT carry out additional checks in response to particular SQLCODES. For example, by embedding SQL SELECT statements in the member you could respond to a - 204 SQLCODE by carrying out further interrogations of your relational environment.

The HELP text you have created is not displayed in response to SQL statements submitted with the ISQL command. You can find out what the predefined SQL/DS HELP text is by entering an ISQL command including the HELP keyword.

7 Importing From DB2

This chapter includes these sections:

Introduction 149
Naming Guidelines 152
Documenting Columns 155

Tailoring Import 157
Tailorable Corporate Executive Routines 158
Global Variables Used in the Import Commands 160

Introduction

You can import information about DB2 objects from the DB2 catalog into the Manager Products environment, using import from DB2 functions. These functions are provided by the import from DB2 panels (menu I31000) or the EXTRACT, RECONCILE, PREVIEW, and POPULATE commands.

DB2 controls access to the information in the catalog. To import information into your Manager Products environment:

- You must have DB2 on the same CPU and running under the same operating system as Manager Products
- Your environment must have access to the DB2 catalog.

Note:

When initially establishing access to your DB2 environment, you can choose a specific DB2 subsystem or plan for your current session.

Refer to "Setting Suffixes Applied to Indicator Array Names" on page 119 for details of accessing a specific DB2 subsystem or plan.

When extracting (see below) from DB2, you can name the creator-owner of the DB2 object you are importing. If you do not specify the creator-owner, your Manager Products Logon Identifier becomes the default.

Importing information from DB2 is in four stages:

Extract. Imports information from the DB2 catalog into Procedures Language Global Variables in the WorkBench Translation Area (WBTA).

Reconcile. Generates proposed member names and types for the DB2 objects, and compares these names and types with any members with matching names and types that exist on the repository. The proposed names include a prefix identifying the type of object they document and, for some types of object, the authorization ID of the object's owner and the database it is stored in.

Preview. Generates complete proposed member definitions and allows the user to inspect them. These members are of the member types provided by DB2 Definition functions. For example, DB2-TABLE members document tables imported from DB2, ITEM members document columns and their data types, MODULE members document DBRMs, and so on.

Populate. Updates the repository with the proposed member definitions.

Refer to the table below for details of the objects that you can import and the default member types and proposed names they are given in the repository.

The systems administrator can tailor the way proposed members are generated from the information on the WBTA. For example, proposed members can be given names and member types which suit your repository standards.

If you import information about an object, information about objects directly related to that object may also be imported. The proposed members documenting the objects will contain attributes defining their relationships to each other.

Manager Products users who have not already documented their DB2 environment in the repository can therefore do so in a short amount of time. Users who have already documented their DB2 environment in the repository can reconcile their existing member definitions with the objects in DB2, to ensure that their documentation is both accurate and complete.

Having documented your DB2 environment in the repository you can use the functions provided by Manager Products to analyze, enhance and maintain that environment:

Table 19 Default Repository Member Types and Names of Imported Objects

DB2 Object	Member Type	Member Name
ALIAS	DB2-ALIAS	AL-*owner-name*
COLUMN	ITEM	IT-*name*
DATABASE	DB2-DATABASE	DB-*name*
DBRM	MODULE	MO- *name*

Table 19 Default Repository Member Types and Names of Imported Objects

DB2 Object	Member Type	Member Name
EDIT PROCEDURE	MODULE	MO-*name*
FIELD PROCEDURE	MODULE	MO-*name*
INDEX	DB2-INDEX	IX-*owner-name*
OWNER	DB2-USER	US-*name*
LOCATION	DB2-LOCATION	LN-*name*
PLAN	DB2-PLAN	PL-*name*
STORAGE GROUP	DB2-STOGROUP	SG-*name*
TABLE	DB2-TABLE	TB-*owner-name*
TABLE SPACE	DB2-TBSPACE	TS-*database-name*
VALIDATION PROCEDURE	MODULE	MO-*name*
VIEW	DB2-VIEW	VW-*owner-name*
PACKAGE	DB2-PACKAGE	PK-*name*
COLLECTION	DB2-COLLECTION	CL-*name*
DISTINCT TYPE	ITEM	IT-*name*
		ALIAS SQL '*name*'

where:

name is the name of the object on the DB2 catalog

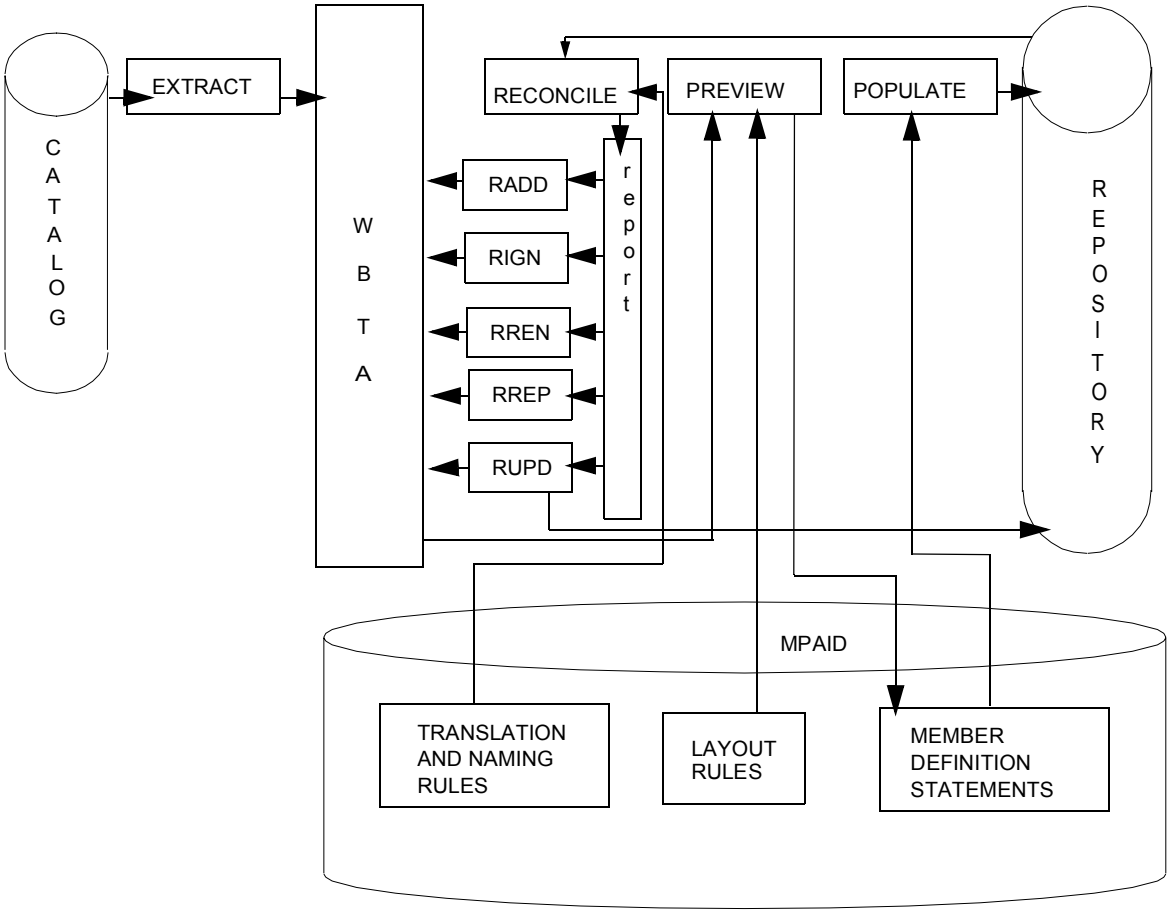
owner is the DB2 authorization ID of the owner of the object

database is the name of the database in which a table space is stored.

Refer to the *ASG-Manager Products Procedures Language* manual for details of Global Variables.

Figure 30 illustrates how information is imported into the Manager Products environment.

Figure 30 • How Information is Imported



Naming Guidelines

You can use the information imported about objects in your relational environment to generate proposed members. Different types of object are documented in particular types of member.

Proposed names are given to the members documenting the external objects. The name of the member is made up of:

- A *prefix* identifying the member and object type
- For certain types of object, the authorization ID of the *owner* of the object in your relational environment
- For table spaces, the *database* they are stored in
- The *name* of the object on the catalog

Each part of the member name is separated by a hyphen.

For example, a proposed member documenting a DB2 table is named TB-owner-name.

Refer to the table below for details of the default naming rules for proposed members documenting imported objects.

The name of the objects as held on the catalog (excluding the authorization ID) is entered in the ALIAS attribute of proposed members. The names of columns as held on the catalog are entered in the KNOWN-AS attributes of proposed members documenting tables.

Underscores in the names of external objects are converted into hyphens when information is imported from the object.

You can rename proposed members using the RECONCILE and RREN commands, or the import from DB2 panels.

You must delimit proposed member names containing characters from the Manager Products extended character set or you will be unable to enter the member's definition into the repository. To find out which characters you can use to delimit member names, use the QUERY STRING-DELIMITER command.

Refer to the *ASG-ControlManager User's Guide* for details of the Manager Products character set and the QUERY command.

If you have already documented your relational environment in the repository, you should ensure that the proposed member's names comply with your existing naming standards.

You cannot accurately reconcile proposed members documenting external objects with existing repository members if the proposed member's names do not comply with your naming standards.

During reconciliation a report comparing the proposed members with any existing repository members which have the same name is displayed.

If reconciliation is inaccurate, objects in your relational environment may be documented in more than one repository member, and the one-to-one correspondence between external objects and repository members may be lost.

There are several points about naming standards to consider, if you have previously created the objects in your relational environment by exporting from repository members.

While you *can* document a column in a GROUP member, imported information about columns are automatically documented in an ITEMs, which have their own default naming prefix. Imported information therefore may not be reconciled with any existing GROUP members documenting the same columns.

When exporting an object, if the object name was generated:

- From the ALIAS or KNOWN-AS attribute of a member
- From a member name longer than that allowed in your relational environment, that was reduced in length by the *Name Reduction Process*
- Using *name editing options*
- With an altered *creator-owner* or *location* prefix

then, on import, its name will not match the name of the repository member documenting that object. In this case, imported information is not reconciled with the original member.

Refer to Appendix A, "Name Reduction Process," on page 511 for details of the Name Reduction Process.

Refer to Chapter 8, "Commands," on page 175 for details of name editing options, and altering the creator-owner or location prefixes.

There are ways of avoiding these problems:

- You can define alternative naming rules which suit your naming standards, for example, to create your own naming prefixes.
- You can tailor interrogation of the repository carried out during reconciliation. For example, if it is your naming standard to generate object names from ALIAS or KNOWN-AS attributes you can use WHOSE ALIAS IS or WHICH HAVE KNOWN-AS commands during reconciliation, to find out if there is an existing member documenting an object. This approach is only effective if you have, as a repository standard, entered unique names in all ALIAS or KNOWN-AS attributes.
- For tables or views, you can specify the full name of repository members (if they do not exceed 30 characters) as the labels of external objects when generating SQL statements for these objects. Labels are not altered by any changes to generated data names, so you can reconcile an object's label with its repository member name when you re-import.

Refer to "Tailorable Corporate Executive Routines" on page 158 for details of how to define your own naming rules.

Imported View Column Naming

In the case of column names derived from imported views, there are occasions where the column name must be derived according to its context. This will typically be where an expression is involved, in which case the repository name for the column is derived according to the data type of the result. A suffix of -EXP is also applied in this case to signify that the name has been derived from an expression. So, for example, a column defined as SUM(SALARY) may result in an extracted column definition of IT-INTEGER-EXP.

There are also two special cases where IMPORT assigns names for VIEW columns. These are:

- Generic selection of all columns
- Columns that are unnamed.

In these cases it is possible to tailor assigned member names in the corporate executive routine MPDY42DFLT by setting variables:

Variable	Description
MPDY_UNNAMED_COLUMN_NAME	Defines a name to be applied where none is found when importing a view.
MPDY_GENERIC_SELECT_NAME	Defines a name to apply for the asterisk expression (*) in a view.

Documenting Columns

ITEM repository members document imported information about columns in your relational environment.

The form-description and USAGE attribute of the ITEM member documents the data type of a column. The default ITEM form-keyword is HELD-AS.

Refer to the table below for details about documenting column data types.

You can document columns in GROUP members, but imported information about columns is always documented in ITEM members.

The systems administrator can tailor how columns and their data types are documented in the repository. Refer to "Tailorable Corporate Executive Routines" on page 158 for details of tailoring how columns are documented.

The following table shows how the data type of a column is documented in the form-description and USAGE attribute of ITEM members:

Table 20 Documenting the Data Type of Imported Columns

Column Data Type	Form-Description	USAGE Attribute
TIME	CHARACTER 8	TIME
DATE	CHARACTER 10	DATE
TIMESTAMP	CHARACTER 26	TIMESTAMP
GRAPHIC(*n*)	CHARACTER *n*	GRAPHIC
VARGRAPHIC(*n*)	CHARACTER 1 TO *n*	GRAPHIC
LONG VARGRAPHIC	CHARACTER 16383	GRAPHIC
SMALLINT	BINARY 4	none
INTEGER	BINARY 9	none
FLOAT(*n*) *n* = 1. .21	FLOATING-POINT 6	none
FLOAT(*n*) *n* = 22. .53	FLOATING-POINT 16	none
DECIMAL(*n*,*m*)	PACKED DECIMAL (*n-m.m*)	none
CHARACTER(*n*)	CHARACTER *n*	none
VARCHAR(*n*)	CHARACTER 1 TO *n*	none
LONG VARCHAR	CHARACTER 32767	none
BLOB/CLOD/DBCLOB	CHARACTER *n* [K|M|G]	BLOB/CLOB/DBCLOB
ROWID	CHARACTER *n*	ROWID

where *n* is an integer.

A column data type of DECIMAL(*n*,*m*) means a total of *n* digits and *m* decimal digits. This converts to a form-description expressed with the number of digits to the left of the decimal point (*n-m*) followed by a period, followed by *m*. For example, a column with a data type of DECIMAL(5,3) is documented in a form-description attribute as PACKED-DECIMAL 2.3.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of ITEM form-description and USAGE attributes.

Tailoring Import

The four stages in the import process are: extract, reconcile, preview, and populate. Refer to "Introduction" on page 149 for details of these stages.

You can tailor the reconcile and preview stages, to customize imported information to suit your own purposes and environment. You can tailor *reconcile* to change the way in which extracted information is compared with information already present in your repository. You can tailor *preview* to generate proposed members in a form that suits your environment. For example, you can omit the ALIAS attribute from members.

Note:

You can also tailor some aspects of the import process at installation time. Refer to the information about Manager Products and external environments in your Manager Products installation manual.

Both the reconcile and preview stages call a series of Corporate executive routines during execution, which you can alter to tailor either stage.

Reconcile calls the following routines, in the given order:

- MPDYWTDFLT (and optionally also MPDYWTMT42)
- MPDYWTCVDT
- MPDYWTRDMR (or any of your own routines you wish to call)
- MPDYWTOCOD
- MPDYWTEXCC

Preview calls the following routines, in the given order:

- MPDYMMCNTL
- Either Corporate executive routines to generate proposed members for particular member types, or your own executive routines
- MPDYMMLOCC

You can call your own executive routines, naming them in the USING clause of the RECONCILE or PREVIEW commands or the import from DB2 panels.

Refer to "Tailorable Corporate Executive Routines" on page 158 for details of the above Corporate executive routines. To print any Corporate executive routine, enter:

```
MP PRINT EXECUTIVE corp-exec-name ;
```

ASG supplies the routines as EXECUTIVE-ROUTINE members in the Manager Products Administration repository.

One way to tailor a Corporate executive routine is to copy it to a USER-MEMBER with the same name, then tailor the USER-MEMBER, leaving the master copy intact. The USER-MEMBER will be called in preference to the Corporate executive routine.

Tailorable Corporate Executive Routines

MPDYWTDFLT sets up the default initialization executive routines which are called during reconciliation, and the default form of the contains attribute for DB2-TABLE and DB2-VIEW member definitions. You can alter form settings, using the variable REC_FORM_DESC.

MPDYWTDFLT also optionally calls the tailorable routine MPDYWTMT42, to establish member type checking during reconciliation. MPDYWTDFLR is called to initialize user-specific COMMON attributes.

MPDYWTCVDT converts SQL data types for columns into ITEM form descriptions and usages. It converts each column data type, setting up a DMR_MEM_DESC variable according to default conversion rules.

Refer to "Documenting DB2 Security Information" on page 96 for details of how to document column data types in the repository.

Refer to the table in "Documenting Columns" on page 155 for details of default conversion rules.

MPDYWTRDMR generates proposed member names and types from extracted object information.

MPDYWTOCOD converts SQL keyword codes, which represent attributes specific to the DB2 objects, into repository member attributes. For example, U, which represents a Unique Index, is converted to UNIQUE.

MPDYWTEXCC extracts default common attributes (such as the DESCRIPTION attribute) from the repository. It is passed 2 parameters:

- &p0: The DMR_MEM_NAME array number of the current member being processed.
- &pl: The DMR_MEM_NAME array number of the first member in a chain of members with the same name.

&p1 equals &p0 when the current member is processed for the first (or only) time. If the two values differ, then the current member (at &p0) has already been processed (at p&l) and the default common attribute variables associated with &p1 can be used for &p0; so, no more DRETRIEVEs are needed.

MPDYMMCNTL reads the type of each proposed member (resulting from reconciling), then passes control to the appropriate Corporate executive routine (listed below) to generate a repository definition for that object type.

DB2 Object	Executive Routine Called	Member Type Generated
alias	MPDYMM12AL	DB2-ALIAS
authorization-ID	MPDYMMLOUS	DB2-USER
column	MPDYMMLOIT	ITEM
database	MPDYMM12DB	DB2-DATABASE
index	MPDYMM12IN	DB2-INDEX
plan	MPDYMM12PL	DB2-PLAN
package	MPDYMM12PK	DB2-PACKAGE
storage group	MPDYMM12ST	DB2-STOGROUP
table	MPDYMM12TB	DB2-TABLE
table space	MPDYMM12TS	DB2-TBSPACE
view	MPDYMM12VW	DB2-VIEW
procedure or function	MPDYMM12PX	DB2-PROCEDURE
trigger	MPDYMM12TG	DB2-TRIGGER
distinct type	MPDYMMLOIT	ITEM

SQL/DS object	Executive Routine Called	Member Type Generated
column	MPDYMMLOIT	ITEM
authorization-ID	MPDYMMLOUS	SQL-USER
table	MPDYMM32TB	SQL-TABLE
view	MPDYMM32VW	SQL-VIEW
dbspace	MPDYMM32DB	SQL-DBSPACE
index	MPDYMM32IN	SQL-INDEX

The Corporate executive routines listed in the previous tables all:

- Generate REPLACE or ADD command statements, followed by the appropriate member definition for the proposed member
- Call the Corporate executive routine MPDYMMLOCC which sets up the default common attributes for the proposed member.

Global Variables Used in the Import Commands

Reconcile and preview both use global variables on the WBTA to control and store the information extracted from the DB2 catalog. These global variables are grouped according to how they are used. The types of variable, and the naming conventions for each variable, are given below:

- Check constraint data variables (EXT_COL_CHK_)
- Fieldproc data variables (EXT_FPR_)
- Database data variables (EXT_DAT_)
- Storage group data variables (EXT_STO_ and EXT_VOL_)
- Plan data variables (EXT_PLA_ and EXT_DEP_)
- Package/collection data variables (EXT_PAC_)
- Table space data variables (EXT_TBS_)
- Column data variables (EXT_COL_)
- Index data variables (EXT_IND_)
- Table data variables (EXT_TAB_)
- View data variables (EXT_VIE_ and EXT_B)
- Miscellaneous data variables (EXT_)
- Generic import variables (EXT_OBJ_)
- Proposed repository member variables (DMR_MEM_)
- Variables for existing repository members (DMR_DIC_)
- Default common attribute variables (DMR_DIC_)
- References from existing repository members (DMR_REF_).

The following sections list the variables in the above groups, the catalog values assigned to them, and how Manager Products uses them (Refer to the IBM reference manuals for the exact meaning and use of the extracted values).

Fieldproc Variables

Variable Name	Source (DB2)/Use
EXT_FPR_CONSTANT	Contains the parameter list given after the FIELDPROC keyword in the statement that created the column.

Database Data Variables

Variable Name	Source (DB2)/Use
EXT_DAT_BPOOL	sysdatabase.bpool.
EXT_DAT_CCSID	Default encoding scheme for the database. It may be EBCDIC or ASCII.
EXT_DAT_CREATOR_PTR	Pointer to the EXT_OBJ_NAME array that names the creator of the database.
EXT_DAT_CREATORS	sysdatabase.creator
EXT_DAT_GROUP_MEMBER	sysdatabase.group_member
EXT_DAT_STOGROUP	sysdatabase.stgroup
EXT_DAT_STOGROUP_PTR	Pointer to the EXT_OBJ_NAME array that names the storage group used for the database.
EXT_DAT_TYPE	sysdatabase.type

Storage Group Data Variables

Variable Name	Source (DB2)/Use
EXT_STO_CREATOR	sysstogroup.creator
EXT_STO_VCATNAME	sysstogroup.vcatname
EXT_STO_VOL_OCC	Count (distinct sysvolumes.volid).
EXT_STO_VOL_PTR	Pointer to the EXT_STO_VOL_ID array.
EXT_STO_VPASSWORD	sysstogroup.vpassword

Variable Name	Source (DB2)/Use
EXT_VOL_ID	sysvolumes.volid
EXT_VOL_SCREATOR	sysvolumes.sgcreator
EXT_VOL_SNAME	sysvolumes.sgname

Plan Data Variables

Variable Name	Source (DB2)/Use
EXT_DEP_PCNAME	sysplan.creator
EXT_DEP_PNAME	sysplan.name
EXT_PLA_ACQUIRE	sysplan.acquire
EXT_PLA_CREATOR	sysplan.creator
EXT_PLA_DEGREE	sysplan.degree
EXT_PLA_DEP_OCC	Count (distinct.sysdbrm.name).
EXT_PLA_DEP_PTR	Pointer to the EXT_OBJ_NAME array for a dependent DBRM.
EXT_PLA_DISCONNECT	sysplan.disconnect
EXT_PLA_DYNAMICRULES	sysplan.dynamicrules
EXT_PLA_EXPLAIN	plan.explan
EXT_PLA_ISOLATION	sysplan.isolation
EXT_PLA_KEEPDYNAMIC	Indicates whether prepared dynamic statements are to be purged at each commit point.
EXT_PLA_RELEASE	sysplan.release
EXT_PLA_REOPTVAR	Indicates whether the access path is determined again at execution time using input variable values.
EXT_PLA_SQLRULES	sysplan.sqlrules
EXT_PLA_VALIDATE	sysplan.validate

Table Space Data Variables

Variable Name	Source (DB2)/Use
EXT_TBS_BPOOL	systablespace.bpool
EXT_TBS_CCSID	Default encoding scheme for the table space. It may be EBCDIC or ASCII.
EXT_TBS_CLOSE	systablespace.closerule
EXT_TBS_COMPRESS	stablepart.compress
EXT_TBS_CREATOR	systablespace.creator
EXT_TBS_CREATOR_PTR	Pointer to the EXT_OBJ_NAME array that names the creator of the table space.
EXT_TBS_DATABASE	systablespace.dbname
EXT_TBS_DATABASE_PTR	Pointer to the EXT_OBJ_NAME array that names the database of the table space.
EXT_TBS_DSETPASS	systablespace.dsetpass
EXT_TBS_ERASE	systablespace.eraserule
EXT_TBS_FREEPAGE	systablepart.freepage
EXT_TBS_GBPCACHE	systablepart.gpbcache
EXT_TBS_LARGE	systablespace.type
EXT_TBS_LOCKMAX	systablespace.lockmax
EXT_TBS_LOCKPART	systablespace.lockpart
EXT_TBS_LOCKSIZE	systablespace.lockrule
EXT_TBS_MAXROWS	The maximum number of rows that DB2 will place on a data page.
EXT_TBS_NAME	systablespace.name
EXT_TBS_PARTNO	systablepart.partition
EXT_TBS_PCTFREE	systablepart.pctfree
EXT_TBS_PQTY	systablepart.pqty
EXT_TBS_PTN_OCC	systablespace.partition
EXT_TBS_PTN_PTR	Pointer to a partition in the table space.
EXT_TBS_SEGSIZE	null (V1.3) : systablespace.segsize (V2.1)
EXT_TBS_SQTY	systablepart.sqty

Variable Name	Source (DB2)/Use
EXT_TBS_STOGROUP_PTR	Pointer to EXTT_OBJ_NAME that names the storage group used for the tablespace.
EXT_TBS_STOGRP	systablepart.storname
EXT_TBS_VCAT	systablepart.vcatname

Column Data Variables

Variable Name	Source (DB2)/Use
EXT_COL_BIT_DATA	syscolumns.foreignkey
EXT_COL_CHK_PTR	Pointer to the EXT_OBJ_NAME array that names the check constraint for the column.
EXT_COL_COMMENT	syscolumns.remarks
EXT_COL_CORREL	Identifier designating the correlation name of the table or view to which the column belongs (VIEWS ONLY).
EXT_COL_CREATOR	syscolumns.tbcreator
EXT_COL_DEFAULT	syscolumns.default
EXT_COL_DEFAULTVAL	syscolumns.defaultvalue
EXT_COL_EXPRESSION	If the column consists of an expression referring to one or more other columns, this variable contains the expression (VIEWS ONLY).
EXT_COL_FLDPROC	syscolumns.fldproc
EXT_COL_FLDPROC_PTR	Pointer to the EXT_OBJ_NAME array, naming the field procedure for this column.
EXT_COL_GROUPBY	Contains GROUP-BY if the column is used in a GROUP BY attribute in a VIEW; otherwise it contains null (VIEWS ONLY).
EXT_COL_LABEL	syscolumns.label
EXT_COL_LENGTH	syscolumns.length
EXT_COL_NAME	Contains the name of the column.
EXT_COL_NULLS	syscolumns.nulls
EXT_COL_NUMBER	syscolumns.colno
EXT_COL_PKEY	Contains PRIMARY-KEY if the column is a primary key.

Variable Name	Source (DB2)/Use
EXT_COL_PRECISION	syscolumns.(length-scale)
EXT_COL_PSEQUENCE	syscolumns.keyseq
EXT_COL_SCALE	syscolumns.keyseq
EXT_COL_SEQUENCE	syskeys.ordering
EXT_COL_TNAME	syscolumns.tbname
EXT_COL_TYPE	syscolumns.coltype

Index Data Variables

Variable Name	Source (DB2)/Use
EXT_IND_BUFPOOL	sysindexes.bpool
EXT_IND_CLOSE	sysindexes.closernle
EXT_IND_CLUSTER	sysindexes.clustering
EXT_IND_COL_ICREATOR	sysindexes.creator
EXT_IND_COL_INAME	sysindexes.name
EXT_IND_COLUMN_OCC	sysindexes.colcount
EXT_IND_COLUMN_PTR	Pointer to the EXT_OBJ_NAME array that names the columns used in the index.
EXT_IND_COLUMN_SEQ	Shows the sequence in which the column entries are indexed. It may be ASCENDING or DESCENDING.
EXT_IND_CREATOR	sysindexes.creator
EXT_IND_CREATOR_PTR	Pointer to the EXT_OBJ_NAME array that names the creator of the index.
EXT_IND_DSETPASS	sysindexes.dsetpass
EXT_IND_ERASE_RULE	sysindexes.eraserule
EXT_IND_INDEX_TYPE	sysindexes.indextype
EXT_IND_SUBPAGE	Contains the number of subpages per page.
EXT_IND_SUBPAGE_SIZE	sysindexes.pgsize
EXT_IND_TABLE_PTR	Pointer to the EXT_OBJ_NAME array that names the indexed table.
EXT_IND_TCREATOR	sysindexes.tbcreator

Variable Name	Source (DB2)/Use
EXT_IND_TNAME	sysindexes.tbname
EXT_IND_TYPE	sysindexes.uniquerule
EXT_PART_FREEPAGE	sysindexpart.freepage
EXT_PART_GBPCACHE	sysindexes.gbpcache
EXT_PART_PCTFREE	sysindexpart.pctfree
EXT_PART_PIECESIZE	Maximum size of a data set storage piece (in K) that will be used for non-partitioned indexes.
EXT_PART_PRIQTY	sysindexpart.pqty
EXT_PART_SECQTY	sysindexpart.secqty
EXT_PART_STORGROUP_PTR	Pointer to the EXT_OBJ_NAME array that names the storage group used for the index.
EXT_PART_STORNAME	sysindexpart.storname
EXT_PART_VCATNAME	sysindexpart.vcatname

Table Data Variables

Variable Name	Source (DB2)/Use
EXT_TAB_AUDIT	systables.auditing
EXT_TAB_CCSID	Default encoding scheme for the tables. It may be EBCDIC or ASCII.
EXT_TAB_CHECKNO	systables.checks
EXT_TAB_CLUSTER_TYPE	systables.clustertype
EXT_TAB_COL_FCOL_PTR	Pointer to the EXT_0BJ_NAME array for the foreign key which corresponds to the primary key.
EXT_TAB_COL_OCC	systables.colcount
EXT_TAB_COL_PTR	Pointer to the EXT_OBJ_NAME array. It points to the start of the columns contained in the table.
EXT_TAB_COMMENT	systables.remarks
EXT_TAB_CORREL	Identifier that designates the table or view.
EXT_TAB_CREATOR	systables.creator
EXT_TAB_CREATOR_PTR	Pointer to the EXT_OBJ_NAME array that names the creator of the table.

Variable Name	Source (DB2)/Use
EXT_TAB_DATABASE	Database in which table is stored.
EXT_TAB_EDPROC	systables.edproc
EXT_TAB_EDPROC_PTR	Pointer to the EXT_OBJ_NAME array for the edit procedure.
EXT_TAB_FKEY_CNAME	sysforeigukeys.colname
EXT_TAB_FKEY_CREATOR	sysforeignkeys.creator
EXT_TAB_FKEY_CTNAME	sysforeigukeys.tbname
EXT_TAB_FKEY_DELRULE	sysrels.deleterule
EXT_TAB_FKEY_NAME	sysforeignkeys.relname
EXT_TAB_FKEY_OCC	systables.parents
EXT_TAB_FKEY_PTR	Pointer to the EXT_OBJ_NAME array for the first foreign key.
EXT_TAB_LABEL	systables.label
EXT_TAB_PKEY_PCTFREE	Contains the percentage of space in each index page reserved for later insertion and updates of the primary key.
EXT_TAB_PKEY_TAB_COL_OCC	Contains the number of columns in a foreign key.
EXT_TAB_PKEY_TAB_COL_PTR	Contains a pointer to the DMR_MEM_NAME array, which contains the first column in a foreign key.
EXT_TAB_ROWS	systables.card
EXT_TAB_SPACE	systables.tsname
EXT_TAB_SPACE_PTR	Pointer to the EXT_OBJ_NAME array for the TBSPACE, or DBSPACE, which the table occupies.
EXT_TAB_TSOWNER	systables.tsname
EXT_TAB_VALPROC	systables.valproc
EXT_TAB_VALPROC_PTR	Pointer to the EXT_OBJ_NAME array for the validation procedure.

View Data Variables

Variable Name	Source (DB2)/Use
EXT_BCREATOR	sysviewdep.bcreator
EXT_BNAME	sysviewdep.bname
EXT_BTYPE	sysviewdep.btype
EXT_TAB_SELECT_TYPE	Value of SELECT or SUBSELECT for table entries only.
EXT_VIE_CHECK_OPTION	Specifies whether or not the check option was specified in the CREATE VIEW statement.
EXT_VIE_CNAME	Contains the column name in the view.
EXT_VIE_COMMENT	Contains information about the view, supplied by a user via an SQL COMMENT statement.
EXT_VIE_HAVING	Array which contains all the extracted HAVING attributes.
EXT_VIE_HAVING_OCC	Contains the number of array elements which are named in the HAVING attribute of the AS subselect, in the SQL CREATE VIEW statement.
EXT_VIE_HAVING_PTR	Pointer to the EXT_VIE_HAVING array which contains the HAVING attribute.
EXT_VIE_LABEL	The label of the VIEW as given by a LABEL ON statement.
EXT_VIE_SELECT_TYPE	Shows the type of selection of VIEWs. It may contain either ALL or DISTINCT.
EXT_VIE_TABLE_OCC	Contains the number of TABLEs referred to by this VIEW.
EXT_VIE_TABLE_PTR	Pointer to the TABLE referred to by the VIEW.
EXT_VIE_WHERE	Array which contains all the extracted WHERE attributes.
EXT_VIE_WHERE_OCC	Contains the number of array elements which are named in the WHERE attribute of the AS subselect, in the SQL CREATE VIEW statement.
EXT_VIE_WHERE_PTR	Pointer to the EXT_VIE_WHERE array which contains the WHERE attribute.
EXT_VIEW_TEXT	sysviews.text

Package/Collection Variables

Variable Name	Source (DB2)/Use
EXT_PAC_COLLECTION	syspackage.collid
EXT_PAC_CON_OCC	syspackage.sysentries
EXT_PAC_CONTOKEN	syspackage.contoken
EXT_PAC_CREATOR	syspackage.creator
EXT_PAC_CURRENTDAT	syspaclcage.deferprep
EXT_PAC_DEFERPREPARE	Whether PREPARE processing is deferred until OPEN is executed.
EXT_PAC_DEGREE	syspackage.degree
EXT_PAC_DYNAMICRULES	syspackage.dynamicrules
EXT_PAC_EXPLAIN	syspackage.explain
EXT_PAC_ISOLATION	syspackage.isolation
EXT_PAC_KEEPDYNAMIC	Whether prepared dynamic statements are to be purged at each commit point.
EXT_PAC_QUALIFIER	syspackage.qualifier
EXT_PAC_RELEASE	syspackage.release
EXT_PAC_REOPTVAR	Whether the access path is determined again at execution time using input variable values.
EXT_PAC_SQLERROR	syspackage.sqlerror
EXT_PAC_VALIDATE	syspackage.version
EXT_PAC_VERSION	syspackage.version

Check-constraint Variables

Variable Name	Source (DB2)/Use
EXT_CHK_NAME	syschecks.checkname
EXT_CHK_CONDITION	syschecks.checkcondition

Miscellaneous Data Variables

Variable Name	Source (DB2)/Use
EXT_BCREATOR	sysviewdep.bcreator
EXT_BNAME	sysviewdep.bname
EXT_BTYPE	sysviewdep.btype
EXT_DEP_PCNAME	sysplan.creator
EXT_DEP_PNAME	sysplan.name
EXT_VOL_ID	sysvolumes.volid
EXT_VOL_SCREATOR	sysvolumes.sgcreator
EXT_VOL_SNAME	sysvolumes.sgname

Generic Import Variables

These variables are generic to import from any source. They are used during reconciliation to generate proposed member names and types.

Variable Name	Source (DB2)/Use
EXT_OBJ_CHAIN	Contains information to link a parent and its children. The variable contains the array number of the first child if the object is a parent, of the second child if the object is a first child, and so on. For the last child the value of this variable will be null.
EXT_OBJ_CHAIN_END	For a parent object, this variable contains a pointer to its last child. It is used to build the EXT_OBJ_CHAIN array without going through the whole chain.
EXT_OBJ_ID	Contains the fully qualified name for a parent object, normally EXT_OBJ_NAME prefixed by the object's creator and a full stop.
EXT_OBJ_NAME	One of the following catalog sources, depending on the object being extracted: systables.name, syscolumns.name, sysdatabase.name, sysplan.name, sysdbrm.name, sysstogroup.name, systablespace.name, systablepart.name, sysindexes.name, syskeys.name.
EXT_OBJ_OCC	Contains the total number of parent objects extracted.
EXT_OBJ_PARENT_ POINTER	For children this will be the array element number for the EXT_OBJ_NAME array that names the parent object. For parents this will be null.

Variable Name	Source (DB2)/Use
EXT_OBJ_TYPE	One of the following literal values, depending on the object being extracted: TABLE or VIEW, COLUMN, DATABASE, PLAN, PROGRAM, STOGROUP, TBSPACE, INDEX.
EXT_SOURCE	Contains the name of the source database. It can have the value DB2, SQL/DS, or EXF (for external files).

Note:

Other EXT_OBJ_*xx* variables are reserved for future use.

Proposed Repository Member Variables

These are generated during reconciliation and describe the proposed member names and types.

Variable Name	Use
DMR_MEM_CHAIN	Contains a forward pointer for all proposed members with the same name. Note that the last member in the chain will point to the first member.
DMR_MEM_CHAIN_PREV	Contains a backward pointer for all proposed members with the same name. Note that the first member in the chain will point to the last member.
DMR_MEM_DESC	Contains a proposed ITEM member's form description.
DMR_MEM_FUNC	Defines the repository update function to be applied to the object. It may contain REPLACE, ADD, IGNORE, or AMEND.
DMR_MEM_GEN	Indicator used during preview, showing whether or not to generate the definition for the object. It may contain either GEN, NOGEN, or REF. GEN means generate the definition. NOGEN means do not generate the definition because: a) the parent of this object has DMR_MEM_FUNC set to IGNORE, or b) the definition will be generated via another object in this array because another parent refers to the same member. REF means do not generate the definition because the member is a referenced object.

Variable Name	Use
DMR_MEM_NAME	Contains the proposed member name for an object, which is created by the default naming rule (MPDYWTRDMR) and optionally by a user-supplied naming rule.
DMR_MEM_TYPE	Contains the repository data type for the object; for example, DB2-TABLE or ITEM.
DMR_MEM_VERSION	Contains the proposed version number for a generated ITEM. If no versions exist on the repository this will be a null.

Variables for Existing Repository Members

These are populated when a proposed member already exists on the repository.

Variable Name	Use
DMR_DIC_COND	Defines the condition of the member in the repository, as with the CONDITION column seen after a LIST command; for example, SCE ENC, or *SCE DUM.
DMR_DIC_MATCH	Shows whether a proposed member's form description matches a form description for a version of that member in the repository. If there is no match, the indicator is set to zero. If there is a match, the indicator shows the character position in the repository form description at which the match begins. For example, if the repository form description contains HELD-AS CHARACTERS 8 and the proposed form description is CHARACTERS 8 then DMR_DIC_MATCH will be set to 9, because the match begins with the word CHARACTERS which starts at character position 9 in the string HELD-AS CHARACTERS 8.
DMR_DIC_TYPE	Contains the member type of a repository member.
DMR_DIC_VER_OCC	Contains the number of versions if the member is held on the repository as an ITEM. It contains 1 if no versions are specified.
DMR_DIC_VER_FORM	Contains the full form description for a version of an ITEM held on the repository.
DMR_DIC_VER_PTR	Contains a pointer to the DMR_DIC_VER_FORM array if the member is held on the repository as an ITEM.

Common Clause Variables

The number of variables set up depends on whether or not the proposed member already exists in the repository, as most of the variables are populated from information held in the repository member.

If the proposed member is not in the repository, only the NOTE (containing the date and a time stamp) and ALIAS attributes are set up.

If the proposed member is already in the repository, the NOTE attribute is created, (updated to include a message and latest time stamp), and other attributes, if they exist in the repository member, are set up as shown below.

Note:

If you use the NO-COMMON-CLAUSES keyword during reconciliation, only the common clause variables relating to the NOTE and ALIAS attributes (that is, the DMR_DIC_ALI_ and DMR_DIC_NOT_ attributes) are set up.

Variable Name	Use
DMR_DIC_ADM_OCC	Contains the number of lines of ADMINISTRATIVE-DATA held on the repository.
DMR_DIC_ADM_TEXT	Contains the ADMINISTRATIVE-DATA text for the member.
DMR_DIC_ADM_PTR	Pointer to the DMR_DIC_ADM_TEXT array.
DMR_DIC_ALI_OCC	Contains the number of ALIASES for the repository.
DMR_DIC_ALI_NAME	Contains the name of the ALIAS for the member.
DMR_DIC_ALI_TYPE	Contains the type of the ALIAS for the member.
DMR_DIC_ALI_PTR	Pointer to the DMR_DIC_ALT_NAME array.
DMR_DIC_CAT_OCC	Contains the number of lines of CATALOG data held on the repository.
DMR_DIC_CAT_TEXT	Contains the CATALOG data for the member.
DMR_DIC_CAT_PTR	Pointer to the DMR_DIC_CAT_TEXT array.
DMR_DIC_COM_OCC	Contains the number of lines of COMMENT data held on the repository.
DMR_DIC_COM_TEXT	Contains the COMMENT data for the member.
DMR_DIC_COM_PTR	Pointer to the DMR_DIC_COM_TEXT array.
DMR_DIC_DES_OCC	Contains the number of lines of DESCRIPTION data held on the repository.
DMR_DIC_DES_TEXT	Contains the DESCRIPTION data for the member.

Variable Name	Use
DMR_DIC_DES_PTR	Pointer to the DMR_DIC_DES_TEXT array.
DMR_DIC_NOT_OCC	Contains the number of lines of NOTE text held on the repository.
DMR_DIC_NOT_TEXT	Contains the NOTE text for the member.
DMR_DIC_NOT_PTR	Pointer to the DMR_DIC_NOT_TEXT array.

References from Existing Repository Members

These variables are set up if a proposed member already exists on the repository and the repository member refers to other members.

Variable Name	Use
DMR_REF_OCC	Contains the number of members referenced.
DMR_REF_PTR	Pointer to DMR_REF_NAME array.
DMR_REF_MEM_NAME	Contains the name of a member which is referenced.
DMR_ REF_MEM_TYPE	Contains the type of the referenced member.
DMR_REF_MEM_VERSION	Contains the version number of the referenced member.
DMR_REF_RELATIONSHIP	Contains the relationship between the referenced member and a parent.

8 Commands

This chapter includes these sections:

Command Descriptions 176
DB2 ALTER 176
DB2 BIND and DB2 REBIND 195
DB2 COMMENT and DB2 LABEL. 206
DB2 CREATE. 212
DB2 DEBUG 219
DB2 DECLARE 223
DB2 DROP 228
DB2 GRANT and DB2 REVOKE 234
DB2 LIST CYCLES 239
DB2 LIST TABLES 240
DB2 PLOT CLUSTER 242
DB2 PLOT REFERENTIAL-STRUCTURES 245
DB2 POPULATE 250
DB2 PREVIEW 262
DB2 PRODUCE 273
DB2 RECALCULATE 282
DB2-RENAME. 285
DB2 REPORT. 288
DB2 SIZE 291
DB2 SYNONYM 295
EXTRACT DB2 299
ISQL 306
POPULATE 309
PREVIEW IMPORT. 312
RADD 316
RECONCILE 317
RIGN. 329
RREN 330
RREP. 331
RUPD 332

Output Generation Options 334
Sending Generated Output to a USER-MEMBER 334
Sending Generated Output to a Sequential Dataset 335
Sending Generated Output to a Partitioned Dataset 336
Sending Generated Output to PRINT 336

Examples of Output Generation Options 336

Name Editing Options. 337
Dropping or Replacing a Name. 338
Inserting a Character String Within a Name 338
Examples of Name Editing Options 339

Command Descriptions

This section describes the commands provided by Manager Products to support your DB2 environment. The commands are documented in alphabetical order of command name.

DB2 ALTER

DB2 ALTER generates one or more SQL ALTER TABLE, INDEX, TBSPACE, STOGROUP, FUNCTION, PROCEDURE, TRIGGER, or TYPE statements from the definition of a DB2-TABLE, DB2-INDEX, DB2-TBSPACE, DB2-STOGROUP, DB2-PROCEDURE, DB2-TRIGGER, or ITEM member.

Refer to "DB2 ALTER Syntax" on page 187 for the syntax of the DB2 ALTER command.

You can generate SQL ALTER statements from the following repository member types:

- DB2-INDEX
- DB2-FUNCTION or DB2-PROCEDURE
- DB2-STOGROUP
- DB2-TABLE
- DB2-TBSPACE

and then apply these SQL statements in your DB2 environment to alter specified DB2 objects. By accurately documenting the DB2 objects with repository member definitions, you can maintain these objects with SQL statements generated from the definitions.

You can combine any of the options available for each member type, in any way you wish. This means that you can generate several SQL ALTER statements to make a combination of alterations to an object in your DB2 environment, using one DB2 ALTER command and one member definition.

The DB2 ALTER command does not change attributes in the relevant member. You should add or modify these attributes before entering the command. If the attributes are not present in the member definition, no SQL statements are generated. If they are present but have not been modified, the SQL statement(s) generated may be rejected when applied to your DB2 environment.

Note:

You may generate an SQL ALTER TABLE statement to drop a referential constraint, validation routine, or primary key. If so, you should drop the attributes from which the SQL statement will be generated after entering the DB2 ALTER command. If the attributes have already been removed the SQL statement will not be generated.

If you use distributed databases, you can specify a table using a location as part of that table's name, to uniquely identify it across multiple sites.

Generated statements are displayed on the screen. You can tailor this output, file it on the MP-AID, and/or send it to an external dataset, using output generation options. You can also tailor output by calling executive routine (user exit routines) at set points (user exits) during output.

The systems administrator can tailor output by altering the DB2 Profile.

Refer to "Output Generation Options" on page 334 for details of output generation options.

Refer to "Tailoring Output" on page 109 for details of tailoring output.

Adding Columns to a Table

To generate SQL ALTER TABLE statements to add columns to a table, enter:

```
DB2 ALTER member ADD COLUMNS n;;
```

where:

member is the name of a DB2-TABLE repository member

n is a number from 1 to 299 specifying the columns to be added to the table. The columns to be added to a table are the last *n* columns derived from the COLUMNS attribute of the DB2-TABLE member from which the SQL ALTER TABLE statement is being generated. *n* must be less than the total number of columns derived from the COLUMNS attribute.

A separate SQL ALTER TABLE statement is generated for each column to be added to a table.

An SQL statement to add a column to a table will be rejected when applied to your DB2 environment if the column already exists in the table.

You should therefore ensure that the ITEMs and GROUPs, which define the new columns to be added to a table, are specified in the COLUMNS attribute after the members which define the existing columns in a table. Existing columns should not be included in the n columns to be added to a table.

The DB2 data type of columns generated in SQL ALTER TABLE statements is derived from the definition of the ITEMs and GROUPs specified in the COLUMNS attribute.

DB2 requires that new columns added to a table must allow a null or default value. The DB2 ALTER command therefore displays a warning message if any of the columns are defined in the DB2-TABLE member definition as being NOT-NULL. An SQL ALTER TABLE statement to add columns to a table cannot be generated from a DB2-TABLE member that contains an EDITPROC clause.

Refer to Chapter 4, "Repository Definition," on page 91 for details of generating column data types.

Adding or Dropping Referential Constraints on a Table

To generate an SQL ALTER TABLE statement to add a referential constraint to a table, enter:

```
DB2 ALTER member ADD CONSTRAINT NAMED constraint-name;
```

or

```
DB2 ALTER member ADD CONSTRAINT NUMBER n;
```

where:

member is the name of a DB2-TABLE repository member

constraint-name is a name specified in the NAMED attribute of the DB2-TABLE member which identifies the referential constraint to be added or dropped

n is a number identifying which referential constraint is to be added or dropped by its sequence among other referential constraints in the DB2-TABLE member definition.

To generate an SQL ALTER TABLE statement to drop a referential constraint from a table, enter:

```
DB2 ALTER member DROP CONSTRAINT NAMED constraint-name;
```

or

```
DB2 ALTER member DROP CONSTRAINT NUMBER n;
```

Referential constraints are defined in the CONSTRAINT attributes of DB2-TABLE members. The SQL ALTER TABLE statement will be generated from a particular CONSTRAINT attribute. A DB2-TABLE member can have any number of CONSTRAINT attributes.

For example, to generate an SQL ALTER TABLE statement to add a referential constraint on a table from the third CONSTRAINT attribute in the member TB-DJB-CUST, enter:

```
DB2 ALTER TB-DJB-CUST ADD CONSTRAINT NUMBER 3;
```

The SQL statement will be rejected when submitted to DB2 if you attempt to add a referential constraint that already exists in the table. The existing referential constraint will not be modified. If you wish to replace it you should use the DB2 ALTER command to first drop the existing referential constraint and then to add the new one.

Adding or Dropping a Primary Key on a Table

To generate an SQL ALTER TABLE statement to add a primary key, enter:

```
DB2 ALTER member ADD PRIMARY-KEY;
```

where *member* is the name of a DB2-TABLE repository member.

To generate an SQL ALTER TABLE statement to drop a primary key, enter:

```
DB2 ALTER member DROP PRIMARY-KEY;
```

Primary key columns are defined with the PRIMARY-KEY attribute in the DB2-TABLE member. All columns in the DB2-TABLE that have an associated PRIMARY-KEY keyword are generated as part of the primary key.

Columns allowing a null value cannot be part of the primary key. The DB2 ALTER command therefore displays a warning message if any of the columns defined in the DB2-TABLE member as being part of the primary key do not have an associated NOT-NULL or NOT-NULL-WITH-DEFAULT keyword.

The SQL statement will be rejected when submitted to DB2 if you attempt to add a primary key to a table that already has one. The existing primary key will not be modified. If you wish to replace it you should first use the DB2 ALTER command to drop the existing primary key and then to add the new one.

Adding, Modifying, or Dropping Unique Keys on a Table

To generate an SQL ALTER TABLE statement to add a unique key to a table, enter:

```
DB2 ALTER member ADD UNIQUE ;
```

where *member* is the name of a DB2-TABLE repository member containing the desired unique keys.

To generate a specific unique key, you must identify the desired key with its constraint name or its index name:

```
DB2 ALTER member ADD UNIQUE UNIQUE-CONSTRAINT name
DB2 ALTER member ADD UNIQUE UNIQUE-INDEX name
```

where *name* is the constraint name or the index name of the unique key to be generated. As in the primary keys of a table, columns allowing nulls cannot be part of the unique keys. However, unlike primary keys, you can define multiple unique keys in a table so long as no two are identical.

Adding, Modifying, or Dropping a Validation Routine on a Table

To generate an SQL ALTER TABLE statement to add or modify a validation routine, enter:

```
DB2 ALTER member ADD VALIDPROC ;
```

where *member* is the name of a DB2-TABLE repository member.

To generate an SQL ALTER TABLE statement to drop a validation routine, enter:

```
DB2 ALTER member DROP VALIDPROC ;
```

The validation routine is defined in and generated from the VALIDPROC attribute of the DB2-TABLE member.

An SQL statement to add a validation routine will, when submitted to DB2, modify any existing validation routine on that table.

Adding, Modifying, or Dropping a Check-Constraint

To generate an SQL ALTER statement to add or modify a check-constraint, enter:

```
DB2 ALTER member ADD CHECK check-constraint-name
```

where:

member is the name of a DB2-TABLE repository member

check-constraint-name is an SQL long identifier.

To generate an SQL ALTER statement to remove a check-constraint, enter:

```
DB2 ALTER member DROP CHECK check-constraint-name
```

Adding or Dropping a Dropping Restriction

To generate an SQL ALTER statement to apply a restriction to the table, preventing it from being dropped in DB2, enter:

```
DB2 ALTER member RESTRICT-ON-DROP
```

where *member* is the name of a DB2-TABLE repository member.

To generate an SQL ALTER statement to remove the dropping restriction from the table, enter:

```
DB2 ALTER member RESTRICT-ON-DROP
```

If the DB2-TABLE definition includes WITH-RESTRICT-ON-DROP then the DB2-ALTER table RESTRICT-ON-DROP will generate an ADD RESTRICT-ON-DROP. If it is missing a DROP will be generated.

Adding or Modifying the Auditing Option on a Table

To generate an SQL ALTER statement to add or modify auditing options, enter:

```
DB2 ALTER table AUDIT ;
```

where *table* is a DB2-TABLE repository member.

Auditing options are defined in and generated from the AUDIT attribute of the DB2-TABLE member. If this attribute is not present, an option of AUDIT NONE is generated.

An SQL ALTER statement to add an auditing option will, when submitted to DB2, modify any existing auditing option on that table.

Note:

If you use the AUDIT keyword with the ADD/DROP options, you must ensure that it immediately precedes or follows these options.

Expanding Nested Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for this member, you should use the same keywords for this command, by entering either:

```
DB2 ALTER table EXPAND alterations ;
```

or:

```
DB2 ALTER table NO-EXPAND alterations s;
```

where:

table is the name of a DB2-TABLE member

alterations specify the SQL ALTER TABLE statements to be generated.

EXPAND attributes in the DB2-TABLE member are used as the default.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for further details about the EXPAND attribute.

Specifying an Owner of a Table

To generate an SQL ALTER TABLE statement for a table, with a specified owner (overriding any owner named in the relevant member), enter:

```
DB2 ALTER table SQLID owner alterations ;
```

where:

table is the name of a DB2-TABLE member

owner is the authorization ID of a specific user. This must be no more than 8 characters long and delimited

alterations specify the alterations to be generated.

Specifying a Location of a Table

To generate an SQL ALTER TABLE statement for a table, with a specified location (overriding any location defined for the relevant member), enter:

```
DB2 ALTER table LOCATION loc-id alterations ;
```

where:

table is the name of a DB2-TABLE repository member

loc-id is a delimited string of up to 16 characters, giving a DB2 location

alterations specify the alterations to be generated

Refer to "Interrogating Your DB2 Dictionary Schema" on page 102 for details of deriving external names.

Refer to "DB2-USER" on page 476 for details of the DB2-USER member type.

Altering a Storage Group

To generate an SQL ALTER STOGROUP statement to alter the password of a storage group, enter:

```
DB2 ALTER stogroup PASSWORD ;
```

where *stogroup* is the name of a DB2-STOGROUP member.

To generate an SQL ALTER STOGROUP statement to add a number of volumes to the start or end of a storage group, enter either:

```
DB2 ALTER stogroup ADD n ;
```

or:

```
DB2 ALTER stogroup ADD LAST n ;
```

where *n* is the number of volumes to be added.

For example, to generate an SQL ALTER STOGROUP statement to add the last 3 volumes listed in the DB2-STOGROUP member SG-O3 to the relevant storage group in your DB2 environment, enter:

```
DB2 ALTER SG-03 ADD LAST 3 ;
```

Similarly, to generate an SQL statement to remove a list of volumes, enter:

```
DB2 ALTER stogroup REMOVE FIRST n ;
```

or:

```
DB2 ALTER stogroup REMOVE LAST n ;
```

where *n* is the number of volumes to be removed, starting either from the front or the back of the volume list in the DB2-STOGROUP definition.

Alternatively, to generate SQL ALTER STOGROUP statements to remove a selection of volumes from a DB2 storage group, enter either:

```
DB2 ALTER stogroup REMOVE NAMES volume-names ;
```

or:

```
DB2 ALTER stogroup REMOVE NUMBERS volume-numbers ;
```

where:

volume-names are volumes named in the DB2-STOGROUP member, separated by commas, and delimited

volume-numbers are positions in the volume list in the DB2-STOGROUP member, separated by commas

Altering an Index or a Tablespace

To generate an SQL ALTER INDEX or ALTER TBSPACE statement, enter:

```
DB2 ALTER member alteration-attributes ;
```

where:

member is the name of a DB2-INDEX or DB2-TBSPACE repository member

alteration-attributes specify attributes in the member. These are keywords, and can be a combination of any of the following, in any order:

- BUFFERPOOL
- CLOSE
- DSETPASS
- ERASE
- FREEPAGE
- LOCKSIZE (for DB2-TBSPACE members only)
- LOCKMAX (for DB2-TBSPACE members only)
- COMPRESS (for DB2-TBSPACE members only)
- PART *partition-number*
- PCTFREE
- PRIQTY
- GBPCACHE

- CONVERT-TYPE (for DB2-INDEX only)
- SECQTY
- USBLOCK.

where *partition-number* is an integer specifying a particular partition.

For example, to generate SQL ALTER INDEX statements to alter bufferpool usage on partition number 5 of an index documented by the member IX-DJB-CUST, enter:

```
DB2 ALTER IX-DJB-CUST BUFFERPOOL PART 5 ;
```

Note:

The USBLOCK keyword corresponds to the member's using-block clause.

Altering a Function or Procedure

To generate an SQL ALTER FUNCTION or ALTER PROCEDURE statement, enter:

```
DB2 ALTER member ;
```

where *member* is the name of a DB2-FUNCTION or DB2-PROCEDURE which contains the attributes to be generated. All attributes defined in the member are generated with this command format.

To generate an ALTER statement containing specific attributes, enter:

```
DB2 ALTER member alteration-attributes ;
```

where *alteration-attributes* are those identified by these keywords and can be entered in any order:

Note:

name1 is the specific name to be used to generate a specific function, and *name2* is the external name to be used to generate an external function or procedure.

- SPECIFIC-NAME *name1*
- EXTERNAL-NAME *name2*
- PARAMETER-STYLE
- DETERMINISTIC
- COLLID
- WLM-ENVIRONMENT
- PARAMETERS
- LANGUAGE
- RESULT-SET
- CONTAINS-SQL, MODIFIES-SQL, or READS-SQL (select any one)
- ASUTIME
- PROGRAM-TYPE

- STAY-RESIDENT
- SECURITY
- COMMIT-ON-RETURN
- CALLED-ON-NULL
- SCRATCHPAD
- DBINFO
- ALLOW-PARALLEL
- RUN-OPTIONS
- SPECIAL-REGISTERS
- EXTERNAL-ACTION
- FINAL-CALL
- CARDINALITY
- STATIC-DISPATCH

The keywords are either the same or similar to the SQL keywords with the same meanings. Some of the above attributes are not compatible, and those in conflict or not defined in the named table are ignored. For example, defining LANGUAGE JAVA to a procedure or function requires PARAMETER-STYLE JAVA. Any other option defined for PARAMETER-STYLE is flagged as an error.

Taking User Exits

To take a user exit, enter:

```
DB2 ALTER member alterations USING exit ;
```

where:

member is the name of a DB2-TABLE, DB2-INDEX, DB2-STOGROUP, or DB2-TBSPACE member

alterations are specifications of the alterations to be generated

exit is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken when you use the DB2 ALTER command. The USING keyword overrides any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Output Generation Options

Use the ONTO keyword to direct your generated output to:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

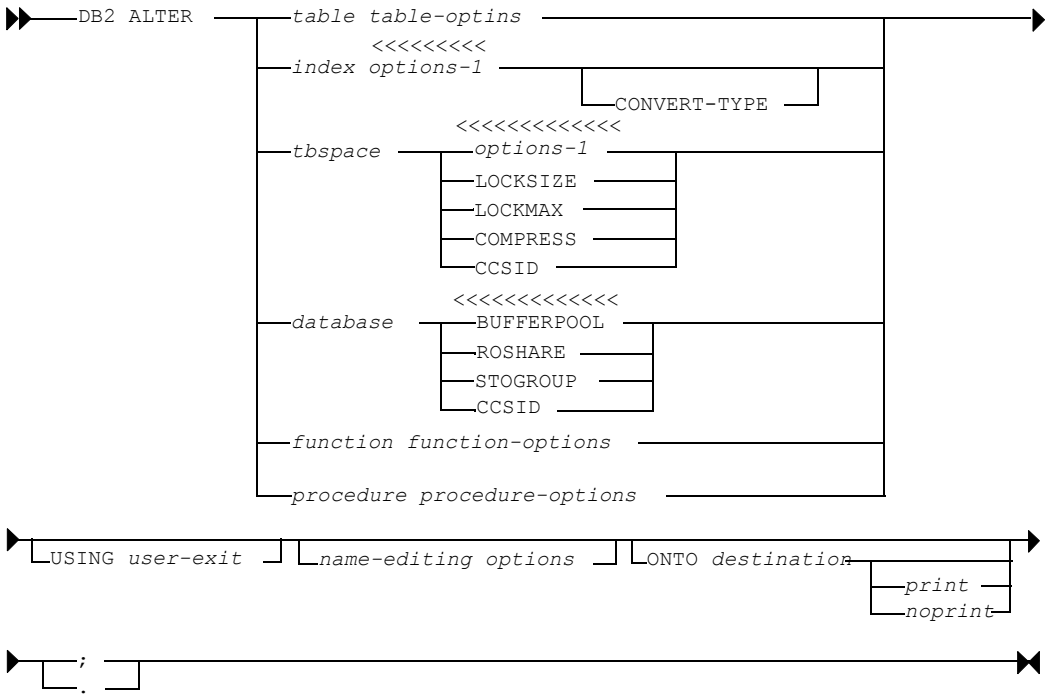
Refer "Output Generation Options" on page 334 for further details of output generation options.

Name Editing Options

You can use the REPLACING/REPLACE, INSERTING, and DROPPING keywords to edit generated data names, before they are output.

Refer to "Output Generation Options" on page 334 for further details of name editing options.

DB2 ALTER Syntax



where:

`CCSID` can be used to generate the CCSID attribute for the tablespace or database named. The CCSID value found in the member is used as the value generated.

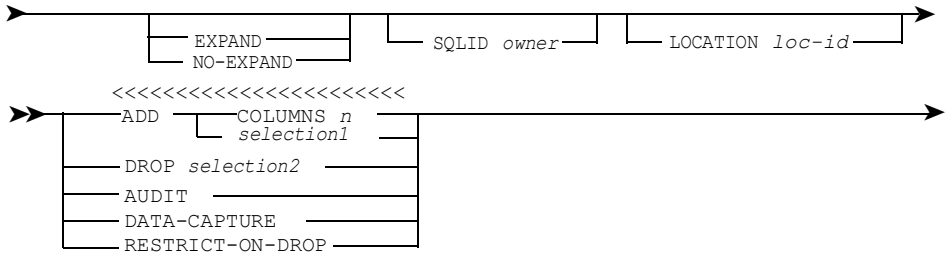
function is the name of a DB2-FUNCTION type member.

procedure is the name of a DB2-PROCEDURE type member.

table, index, tbspace, stogroup, and *database* are names of DB2-TABLE, DB2-INDEX, DB2-TBSPACE, DB2-STOGROUP, and DB2-DATABASE members respectively

user-exit is the name of a user-created executive routine.

table-options are:



Note:

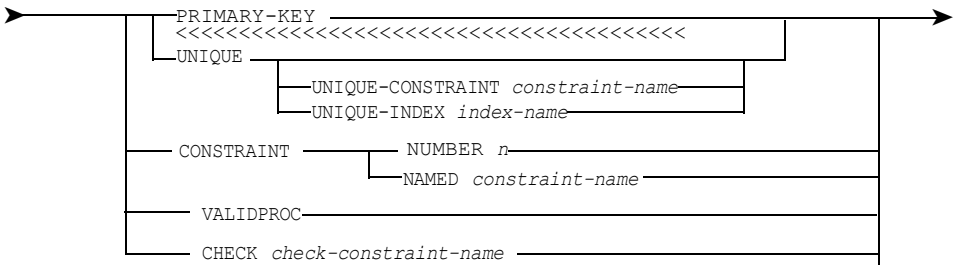
The AUDIT keyword can only be specified once, and must immediately precede or follow all ADD/DROP selections.

where:

owner is a delimited string of up to 8 characters, giving the authorization ID of a particular user.

loc-id is a delimited string of up to 16 characters, giving the name of a DB2 location.

selection1 is:


PRIMARY-KEY
<<<<<<<<<<<<<<<<<<

`UNIQUE-CONSTRAINT` must precede the SQL constraint name of the unique key that is to be altered. A unique key can have a constraint name or an index name.

`UNIQUE-INDEX` must precede the SQL unique index name of the unique key that is to be altered.

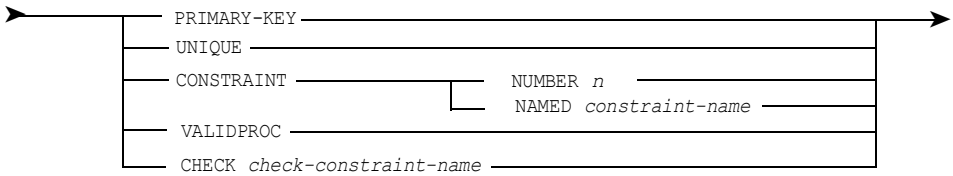
where:

n is an integer

constraint-name is a name specified in the NAMED attribute of a DB2-TABLE member

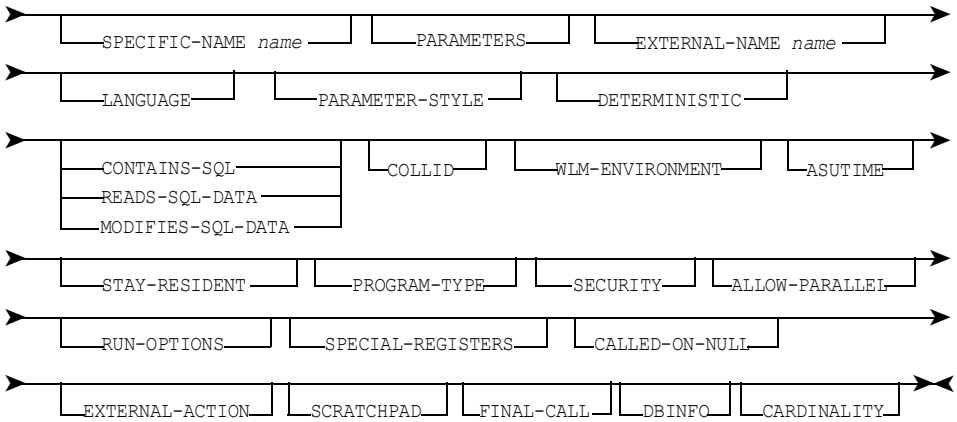
check-constraint-name is an SQL long identifier having up to 18 characters.

selection2 is:

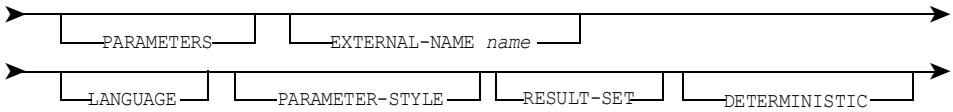


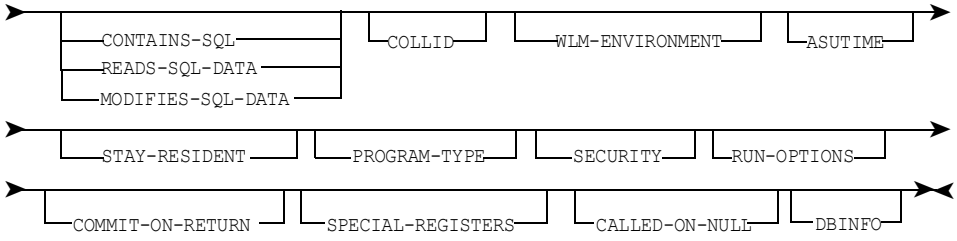
The keyword UNIQUE must be present in order to generate an SQL ALTER DROP UNIQUE statement.

function-options:



procedure-options:





The keywords used for function or procedure options are either identical or similar to the SQL keywords of the same meaning. Except for SPECIFIC-NAME and EXTERNAL-NAME, these keywords are coded to generate that attribute in the SQL ALTER statement using the values in the named function or procedure option.

If any attribute keywords are coded, only those coded are generated. A DB2 ALTER command issued without any attributes listed causes all attributes defined in the named function or procedure option to be generated.

The keywords can be coded in any order, but each keyword can only be coded once. Some keywords are mutually exclusive.

Keyword	Description
SPECIFIC-NAME	Must precede the specific name to be used in generating an SQL ALTER statement for a specific function. If a specific name is found in the named function, it is ignored and the one coded here is used instead. When this attribute is coded, no parameters are generated for the function.
PARAMETERS	Must be present to generate the PARAMETERS attribute in the SQL ALTER statement. It is mutually exclusive with the SPECIFIC-NAME attribute.
EXTERNAL-NAME	Must precede the external name to be used in generating an SQL ALTER statement for an external function or procedure option. If an external name is found in the named function or procedure option, it is ignored and the name coded in either option is used instead.
LANGUAGE	Used to generate the LANGUAGE attribute of the SQL ALTER statement, and identifies the language in which the function or procedure is written. The language specified in the named function or procedure is used as the language value generated. When LANGUAGE is JAVA or COMPJAVA, PARAMETER-STYLE must be JAVA. Also, when LANGUAGE is JAVA, do not specify SCRATCHPAD, FINAL-CALL, DBINFO, or RUN OPTIONS.

Keyword	Description
PARAMETER-STYLE	Generates the parameter style attribute in the SQL ALTER statement generated and identifies the linkage convention used to pass parameters to the function or procedure. The value specified in the named function or procedure is used. If PARAMETER-STYLE is JAVA, the LANGUAGE must be JAVA or COMPJAVA.
RESULT-SET	Generates the result set attribute of the named procedure and specifies the maximum number of query result sets that the procedure can return.
DETERMINISTIC	Generates this attribute of the named function or procedure, and indicates whether it returns the same results on successive calls using the same input. NOT DETERMINISTIC has the synonym VARIANT and means that the same results may not be returned, whereas DETERMINISTIC is the synonym of NOT VARIANT and indicates the same results are returned for same input.
CONTAINS-SQL/ READS-SQL-DATA/ MODIFIES-SQL-DATA	One of these is coded to generate this attribute value in the named function or procedure. The particular keyword coded does not indicate what is generated. If READS-SQL-DATA is coded, but MODIFIES-SQL-DATA is the value in the name function or procedure, MODIFIES-SQL-DATA is generated.
COLLID	Generates this attribute for the named function or procedure, and identifies the package collection that is to be used when the named function or procedure is executed. If no collection ID is found, NO COLLID is generated.
WLM-ENVIRONMENT	Generates this attribute for the named function or procedure, and identifies the MVS Work Load Manager application environment in which the function or procedure is to run.
ASUTIME	Generates this attribute for the named function or procedure, and specifies the total amount of time in CPU service units that a single invocation of this function or procedure can run.
STAY-RESIDENT	Generates this attribute for the named function or procedure, and specifies whether the execution module for the function or procedure remains in memory after it ends.
PROGRAM-TYPE	Generates this attribute for the named function or procedure, and specifies whether the function or procedure runs as a main routine or as a subroutine.
SECURITY	Generates this attribute for the named function or procedure, and identifies the MVS Work Load Manager application
RUN-OPTIONS	Generates this attribute for the named function or procedure, and identifies Language Environment run time options to be used for this function or procedure. This attribute is mutually exclusive with LANGUAGE=JAVA.

Keyword	Description
COMMIT-ON-RETURN	Generates this attribute for the named procedure, and indicates to DB2 whether to commit the transaction immediately upon return from the procedure.
SPECIAL-REGISTERS	Generates this attribute for the named function or procedure. If set to the value INHERIT, the function or procedure inherits the values of the special registers according to the rules set in table of characteristics of special registers. IF set to DEFAULT, the values are set to the special registers default values.
CALLED-ON-NULL	Generates this attribute for the named function or procedure. The values possible are CALLED ON NULL and RETURNS NULL ON NULL INPUT. Synonyms for these keywords are NULL CALL for CALLED-ON-NULL and NOT NULL CALL for RETURNS NULL ON NULL INPUT. This attribute indicates whether the function or procedure is called if any of the input arguments is null. RETURNS NULL ON NULL INPUT is defined only for functions, whereas CALLED ON NULL INPUT is defined for both functions and procedures.
EXTERNAL-ACTION	Generates this attribute for the named function, and specifies whether the function or procedure takes an external action that changes the state of an object that DB2 does not manage.
SCRATCHPAD	Generates this attribute for the named function. This attribute is mutually exclusive with LANGUAGE=JAVA.
FINAL-CALL	Generates this attribute for the named function, and specifies whether a final call is made to the function in order to free up resources. This attribute is mutually exclusive with LANGUAGE=JAVA.
DBINFO	Generates this attribute for the named function or procedure, and specifies whether specific information that DB2 knows is passed to the function or procedure. This attribute is mutually exclusive with LANGUAGE=JAVA.
CARDINALITY	Generates this attribute for the named function, and the value coded for this attribute specifies an estimate of the number of rows that the function returns. The value is used for optimization.

where:

`constraint-name` is an SQL long identifier having up to 18 characters

`check-constraint-name` is an SQL long identifier having up to 18 characters.

options-1 are:

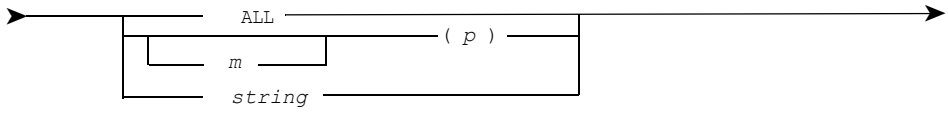
```
    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 ──┬─ BUFFERPOOL ─┬──────────────────────────────────────────────────────►
   ├─ CLOSE ──────┤
   ├─ DSETPASS ───┤
   ├─ PART integer┤
   ├─ FREEPAGE ───┤
   ├─ PCTFREE ────┤
   ├─ USBLOCK ────┤
   ├─ PRIQTY ─────┤
   ├─ SECQTY ─────┤
   ├─ ERASE       ┤
   └─ GBPCACHE ───┘
```

where *integer* is a partition number.

stogroup-options are:

```
 ──┬─ PASSWORD ─┬─┬─ REMOVE ─┬─ NAMES volume ──────────────┬──────────────►
                 │          │      <<,<                    │
                 │          │      << ,<<<                 │
                 │          ├─ NUMBERS position ──────────┤
                 │          ├─ FIRST vol-number ──────────┤
                 │          └─ LAST vol-number ───────────┘
 ──┬─ ADD ─┬─┬─ vol-number ─┬──────────────────────────────────────────────►
       │   │ │              │
       │   │ └─ LAST ───────┘
       │   │
       │   │
       │   │
```

where *selection* is:



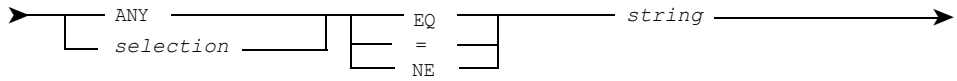
where:

m and *p* are integers in the range 1 to 96.

string is a delimited string of not more than 32 printable characters.

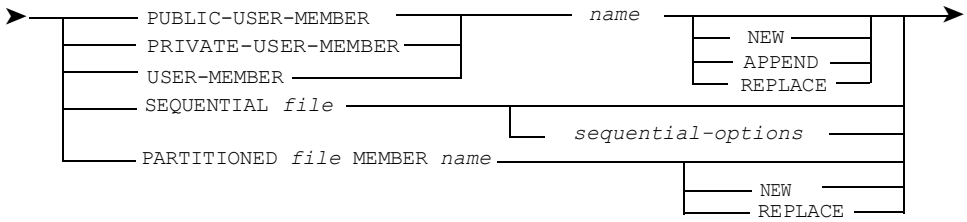
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

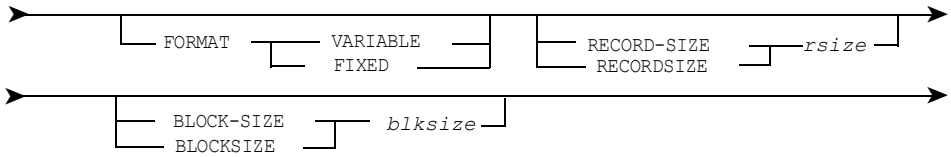


where:

name is the name of a USER-MEMBER.

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length.

blksize is the block size.

DB2 BIND and DB2 REBIND

DB2 BIND and DB2 REBIND generates BIND or REBIND subcommands for your DB2 environment.

Refer to "DB2 BIND PACKAGE Syntax" on page 198 for the syntax of the DB2 BIND and REBIND commands.

You can generate a BIND or REBIND subcommand for the preparation of application plans and packages in your DB2 environment. Plans are documented in DB2-PLAN members and packages in DB2-PACKAGE members.

To generate a BIND or REBIND subcommand, enter:

```
DB2 BIND plan ;
```

or

```
DB2 BIND package ;
```

or

```
DB2 REBIND plan ;
```

or

```
DB2 REBIND package ;
```

where:

plan is the name of a DB2-PLAN member

package is the name of a DB2-PACKAGE member.

Most of the generated output is derived from information held in the DB2-PLAN or DB2-PACKAGE member. Use the DB2 BIND or REBIND commands to specify additional parameters.

The generated output is displayed on the screen. You can tailor this output, file it on the MP-AID, and/or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits*) during output. This process is known as taking user exits.

The systems administrator can tailor output by altering the DB2 profile.

Refer to the DB2-PLAN and DB2-PACKAGE member types in Chapter 9, "Repository Member Types," on page 341 for further details of application plans.

Refer to "Tailoring Output" on page 109 for details of the DB2 profile, and user exits.

Specifying Whether to Keep Dynamic Rules

You can specify whether or not dynamic rules will be kept after the commit point using the KEEPDYNAMIC keyword. To specify that dynamic rules will be kept, enter this command:

```
KEEPDYNAMIC YES
```

If you do not wish to keep the dynamic rules after commit points, enter this command:

```
KEEPDYNAMIC NO
```

Specifying Whether to Determine an Access Path

You can specify whether to determine an access path at run time. To specify that access paths will be determined using default values for input variables, enter this command:

```
VARS NOREOPT
```

To specify that access paths will be determined using the values of input host variables, enter this command:

```
VARS REOPT
```

Controlling Package Creation

When using DB2 BIND you can indicate whether or not package creation should continue in the event of an SQL error. To abandon package creation, enter:

```
SQLERROR NOPACKAGE
```

To allow package creation to continue in error conditions, enter:

```
SQLERROR CONTINUE
```

When using DB2 REBIND for PLAN members, you may indicate that no package is to be created for this plan by entering:

```
NOPKLIST
```

If a package list exists, it is deleted.

Generating Additional Parameters for the BIND or REBIND Command

Use the LIBRARY and ACTION keywords in the DB2 BIND command, or the FLAG keyword in the DB2 BIND or REBIND command, to generate LIBRARY, ACTION, and FLAG parameters for the appropriate subcommand in your DB2 environment.

These keywords have the same meaning as the appropriate parameters in DB2.

Qualifying Unqualified Members

If your plan contains unqualified aliases, indexes, table names, or views, you may provide an implicit qualifier for them by entering:

```
QUALIFIER string
```

where *string* is a string of 1 to 8 characters, delimited by quotes.

Specifying an Owner

To generate a BIND or REBIND command for a DB2-PLAN (or PACKAGE) with a specific owner, overriding any owner already defined for the DB2-PLAN, enter either:

```
DB2 BIND plan SQLID owner ;
```

or

```
DB2 REBIND plan SQLID owner ;
```

where *owner* is a delimited string of up to 8 characters, giving the ID of a specific user.

Taking User Exits

To take a user exit, enter either:

```
DB2 BIND plan USING exit ;
```

or

```
DB2 REBIND plan USING exit ;
```

where *exit* is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken with the DB2 BIND or DB2 REBIND commands. The USING keyword will override any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Name Editing Options

Use the REPLACE/REPLACING, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

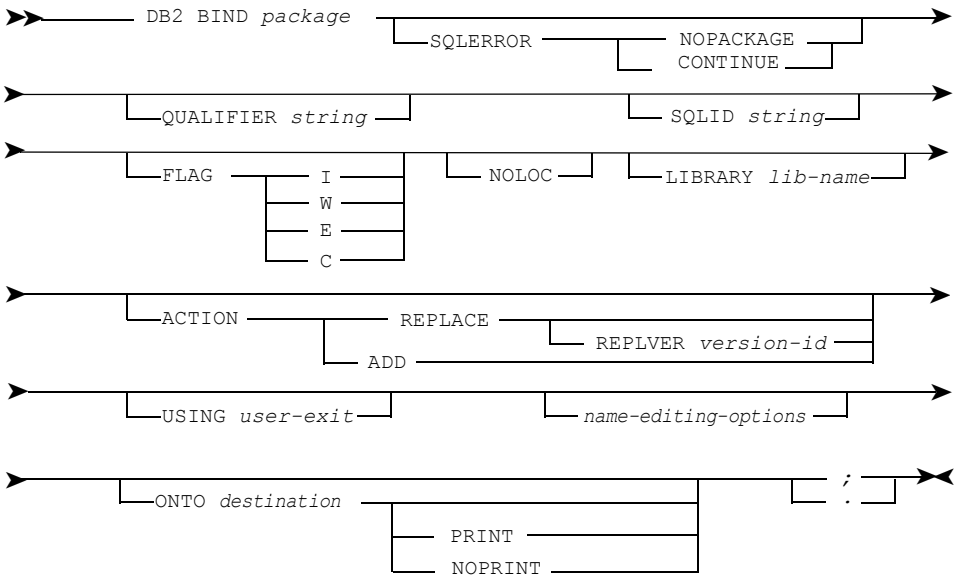
Output Generation Options

Use the ONTO keyword to direct your generated output to a specific destination. This destination can be:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset

Refer to "Output Generation Options" on page 334 for more output generation options.

DB2 BIND PACKAGE Syntax



where:

package is the name of a DB2-PACKAGE repository member

string is a string of 1 to 8 characters, delimited

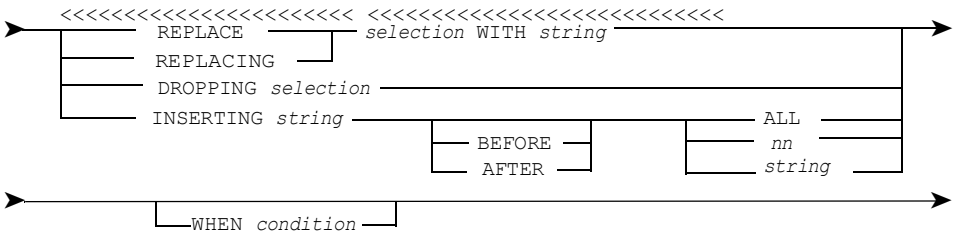
NOLOC specifies that a location name will not be generated as part of the BIND PACKAGE package name. This allows the same package and collection members to be used in the generation of DB2 BIND PLAN where a location may be required.

lib-name is the name, delimited, of a partitioned dataset that contains the DBRM that is being bound into the package

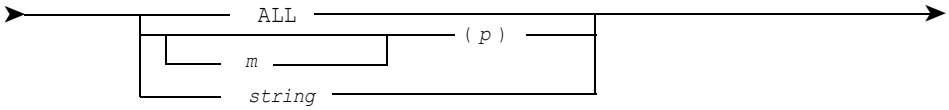
version-id is a string 1 to 64 characters, delimited

user-exit is a delimited string of up to 16 characters, giving the name of an executive routine.

name-editing-options are:



where *selection* is:



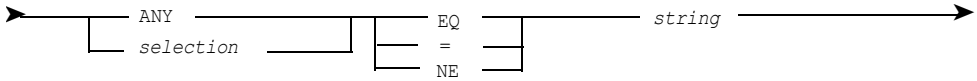
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters

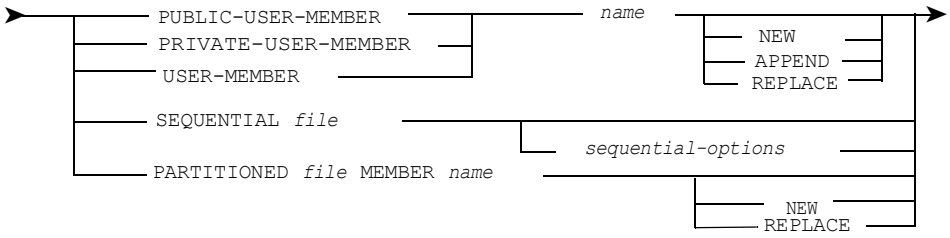
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

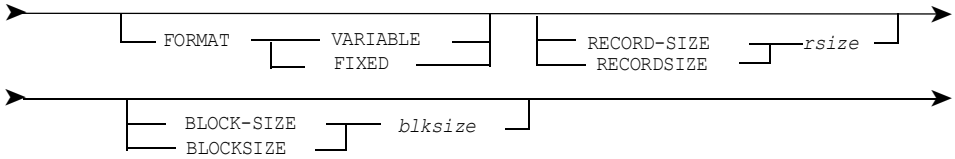


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:

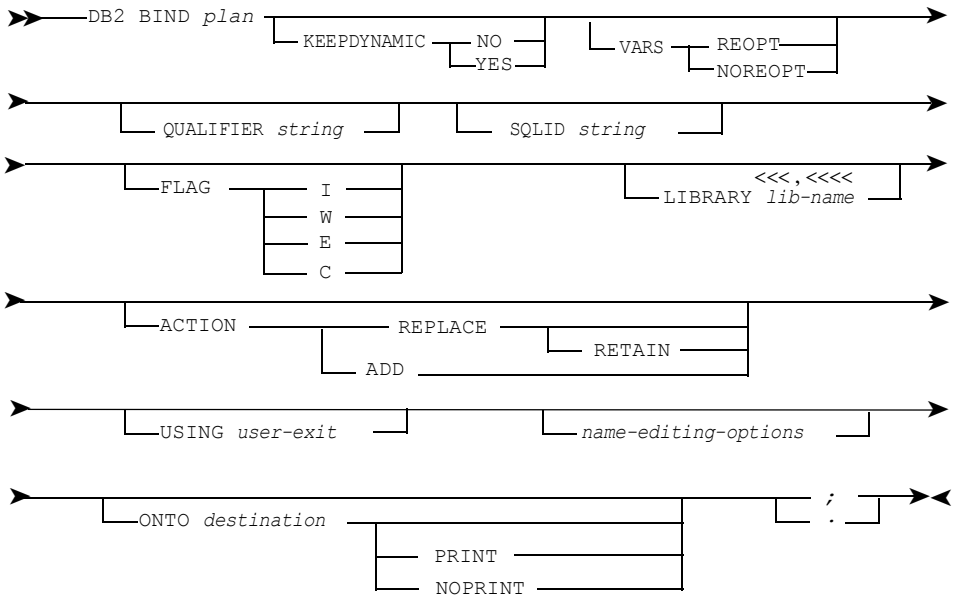


where:

rsize is the record length

blksize is the block size

DB2 BIND PLAN Syntax



where:

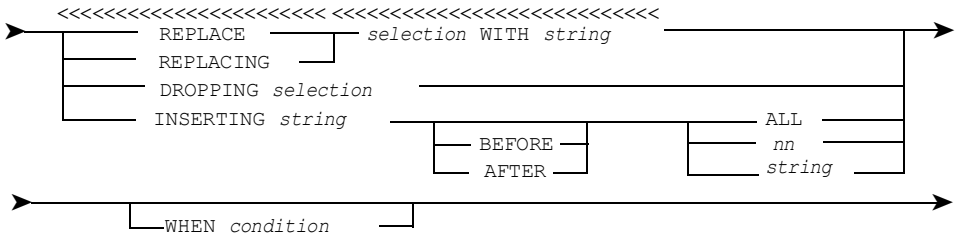
plan is the name of a DB2-PLAN repository member

string is a string of 1 to 8 characters, delimited

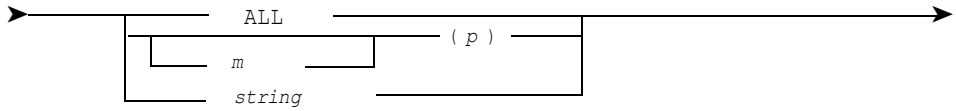
lib-name is the delimited name of one or more partitioned datasets (libraries) that contain as members the DBRMs specified

user-exit is a delimited string of up to 16 characters, giving the name of an executive routine.

name-editing-options are:



where *selection* is:



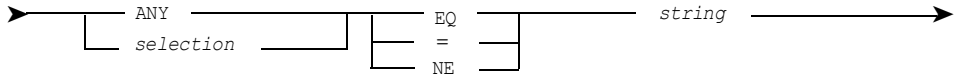
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters

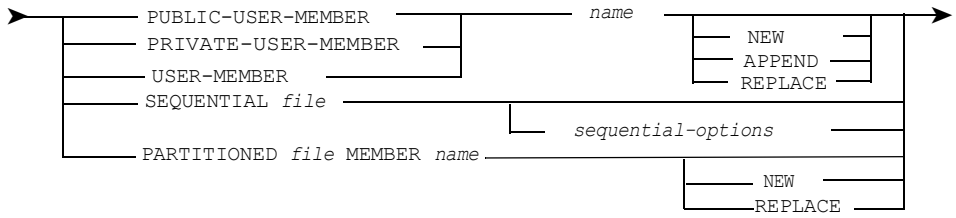
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

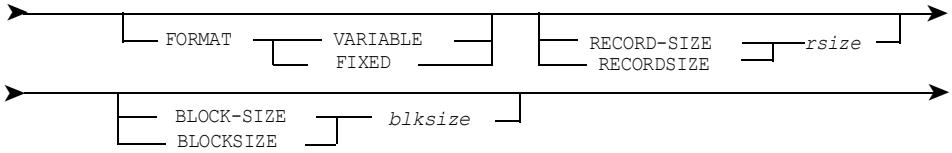


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:

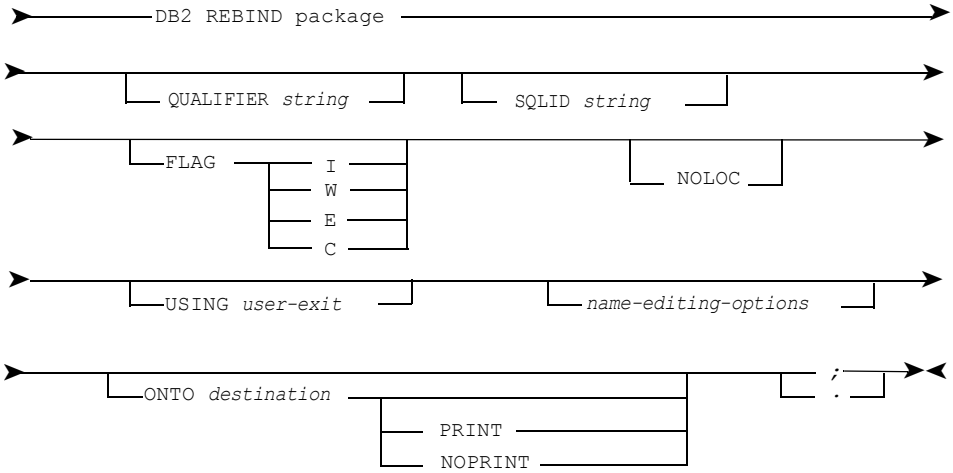


where:

rsize is the record length

blksize is the block size.

DB2 REBIND PACKAGE Syntax



where:

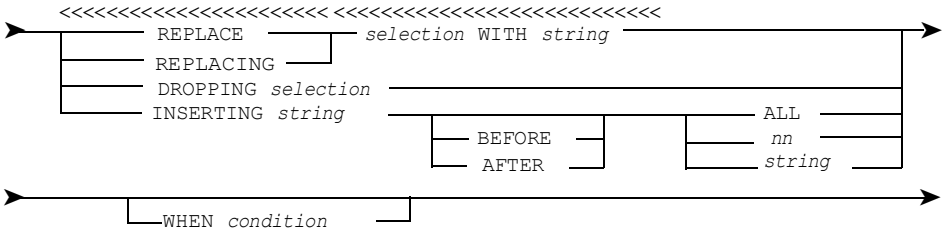
package is the name of a DB2-PACKAGE repository member

string is a string of 1 to 8 characters, delimited

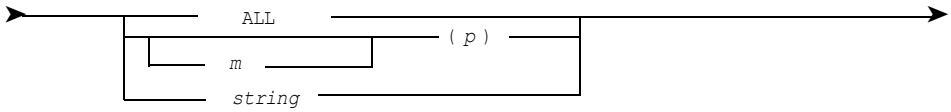
NOLOC specifies that a location name will not be generated as part of the BIND PACKAGE package name. This allows the same package and collection members to be used in the generation of DB2 BIND PLAN where a location may be required.

user-exit is a delimited string of up to 16 characters, giving the name of an executive routine.

name-editing-options are:



where *selection* is:



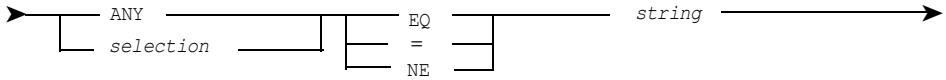
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters

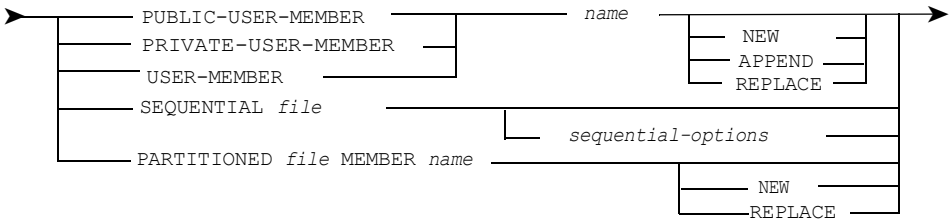
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

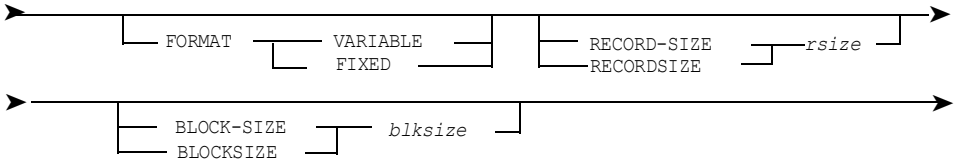


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:

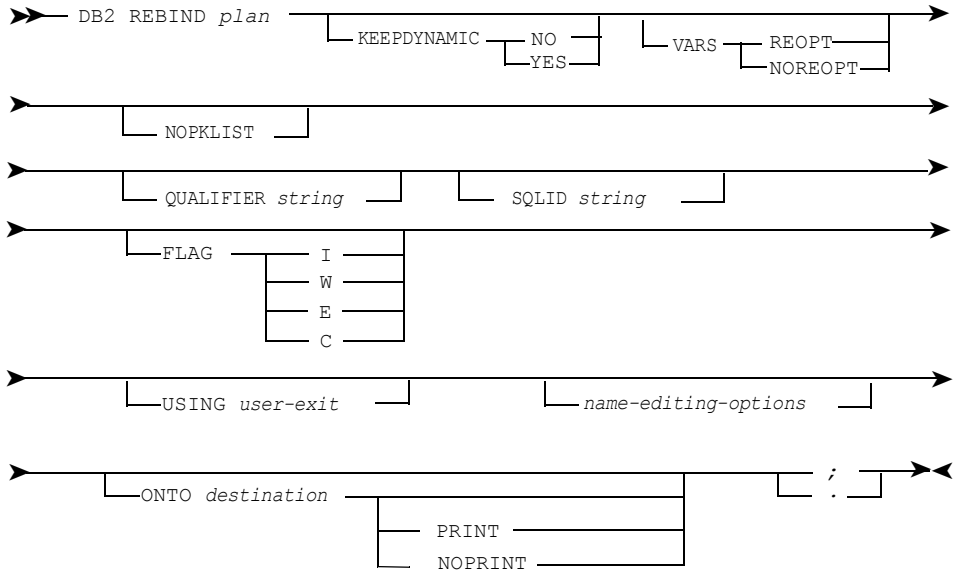


where:

rsize is the record length

blksize is the block size.

DB2 REBIND PLAN Syntax



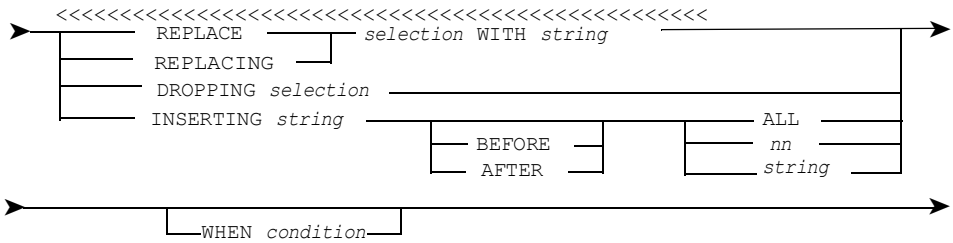
where:

plan is the name of a DB2-PLAN repository member

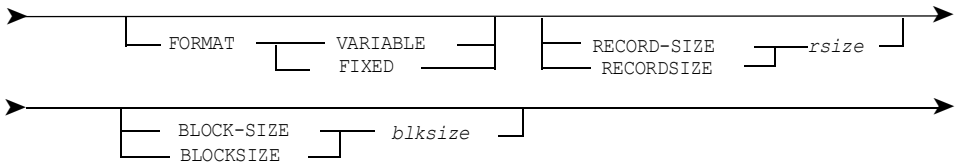
string is a string of 1 to 8 characters, delimited

user-exit is a delimited string of up to 16 characters, giving the name of an executive routine.

name-editing-options are:


<<<

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 COMMENT and DB2 LABEL

DB2 COMMENT and DB2 LABEL generate SQL COMMENT ON or LABEL ON statements from the definition of a DB2-TABLE, DB2-VIEW, or DB2-ALIAS member.

Refer to "DB2 COMMENT and DB2 LABEL Syntax" on page 210 for the syntax of the DB2 COMMENT command.

You can generate SQL COMMENT ON or LABEL ON statements, by entering either:

```
DB2 COMMENT member ;
```

or:

```
DB2 LABEL member ;
```

where *member* is a DB2-TABLE, DB2-VIEW, or DB2-ALIAS repository member for generating labels. If generating comments, *member* can also be distinct-type item, DB2-FUNCTION, DB2-PROCEDURE, DB2-TRIGGER, or DB2-INDEX.

The generated output displays on the screen. You can tailor this output, file it on the MP-AID, or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits*) during output. This process is known as taking user exits.

You can choose to generate only the last *n* comments or labels in a table or view, if you want to generate additional comments or labels after a table or view has been altered and columns added.

You can generate SQL COMMENT ON and LABEL ON statements for a table, view, alias, distinct-type, procedure, trigger, or index, and for columns within a table or view. An SQL COMMENT ON statement is generated from every DB2-COMMENT attribute and an SQL LABEL ON statement from every DB2-LABEL attribute in the definition of the specified DB2-TABLE, DB2-VIEW, or DB2-ALIAS member. If the member does not have a DB2-COMMENT or DB2-LABEL attribute then no statements are generated.

You can also generate comments for distinct-type item, DB2-FUNCTION, DB2-PROCEDURE, DB2-TRIGGER, and DB2-INDEX. There are no options for these member types. The name of the member is the only required input. The member-type named is checked to verify it is one of the valid types, and the SQL COMMENT statements are generated from that member.

Previous to Version 2.5.2, the DB2-COMMENT command was basically the same as the DB2-LABEL command. However, there is no label associated with distinct type, procedure, trigger, and index, so these are not defined to DB2-LABEL, and the two commands are no longer the same. However, previous documentation including DB2-COMMENT and DB2-LABEL is still valid.

Refer to Chapter 9, "Repository Member Types," on page 341 for examples of SQL COMMENT ON and LABEL ON statements generated by the DB2 CREATE command.

Refer to "Tailoring Output" on page 109 for details of tailoring output.

Expanding Nested Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for this member, you should use the same keywords for this command, by entering either:

```
DB2 COMMENT member EXPAND ;
```

or:

```
DB2 LABEL member EXPAND ;
```

or, for NO-EXPAND, either:

```
DB2 COMMENT member NO-EXPAND ;
```

or:

```
DB2 LABEL member NO-EXPAND ;
```

EXPAND attributes in the DB2-TABLE member are used as the default.

Refer to "DB2 CREATE" on page 212 for use of the EXPAND/NO-EXPAND keywords.

Refer to Chapter 9, "Repository Member Types," on page 341 for further details about the EXPAND clause.

Specifying an Owner of a Table or View

To generate SQL COMMENT ON or LABEL ON statements for a table or view with a specified owner (overriding any owner named in the relevant member), enter either:

```
DB2 COMMENT member SQLID owner ;
```

or:

```
DB2 LABEL member SQLID owner ;
```

where *owner* is a delimited string of up to 8 characters, giving the authorization ID of a specific user.

Specifying a Location of a Table or View

To generate SQL COMMENT ON or LABEL ON statements on a table or view, overriding any location given for the relevant member, enter either:

```
DB2 COMMENT member LOCATION loc-id ;
```

or:

```
DB2 LABEL member LOCATION loc-id ;
```

where *loc-id* is a delimited string of up to 16 characters, giving the name of a DB2 location.

Refer to "Interrogating Your DB2 Dictionary Schema" on page 102 for details of deriving external names.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of the DB2-USER member type.

Generating Comments and Labels for the Last Columns in a Table or View

To generate SQL COMMENT ON or LABEL ON statements for only the last *n* columns of a table or view, enter either:

```
DB2 COMMENT member LAST n ;
```

or

```
DB2 LABEL member LAST n ;
```

where *n* is an integer, specifying the number of comments to be generated.

Taking User Exits

To take a user exit, enter either:

```
DB2 COMMENT member USING exit-routine ;
```

or

```
DB2 LABEL member USING exit-routine ;
```

where *exit-routine* is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken when you use the DB2 COMMENT or DB2 LABEL commands. The USING keyword overrides any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Name Editing Options

You can use the REPLACING/REPLACE, INSERTING, and DROPPING keywords to edit data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

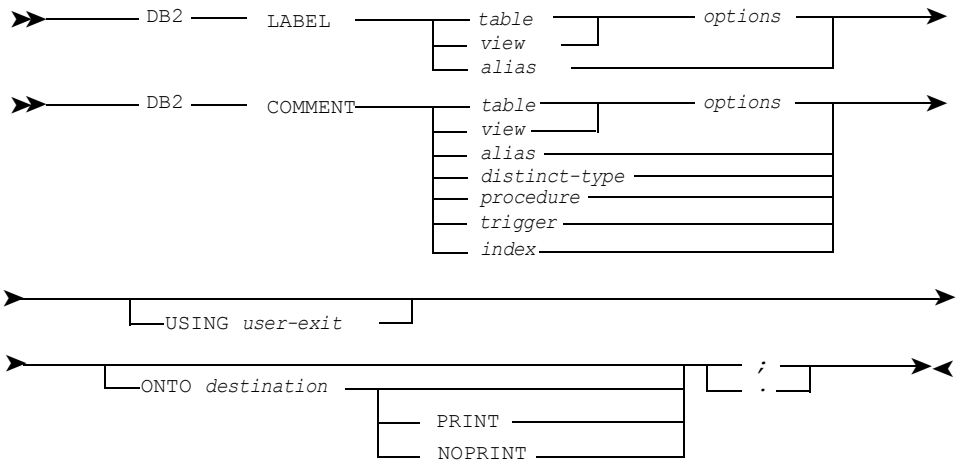
Output Generation Options

You can use the ONTO keyword to direct your generated output to a specific destination. This destination can be:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

Refer to "Output Generation Options" on page 334 for more output generation options.

DB2 COMMENT and DB2 LABEL Syntax



The comment that is generated for the distinct-type item is from the COMMENT attribute of that item. For all other member-types, the comment is taken from the DB2-COMMENT attribute of that member.

where:

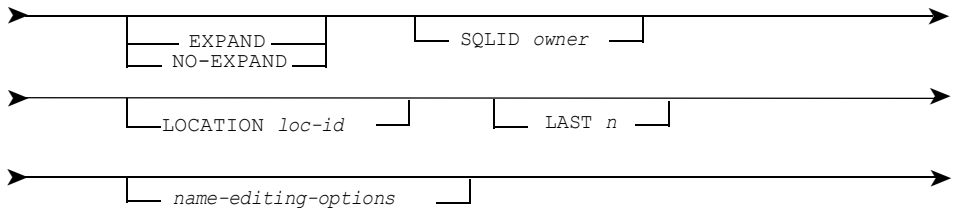
distinct-type is the name of an item in the repository that defines a DB2 distinct type.

table is the name of a DB2-TABLE member

view is the name of a DB2-VIEW member.

alias is the name of a DB2-ALIAS member.

options are:



where:

owner is a delimited string of up to 8 characters, giving the authorization ID of a specific user.

loc-id is a delimited string of up to 16 characters, giving the name of a DB2 location

n is an integer

name-editing-options are:

```
<<<<<<<<<<<<<<<<<<<<<<<< <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 REPLACE          selection WITH string
 REPLACING
 DROPPING selection
 INSERTING string      BEFORE   ALL
                       AFTER    nn
                                string
 WHEN condition
```

where *selection* is:

```
 ALL
 m        ( p )
 string
```

where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters

nn is an unsigned integer in the range 1 to 96.

condition is:

```
 ANY        EQ        string
 selection  =
            NE
```

where *selection* and *string* are as defined above.

user-exit is the name of an executive routine.

destination is:

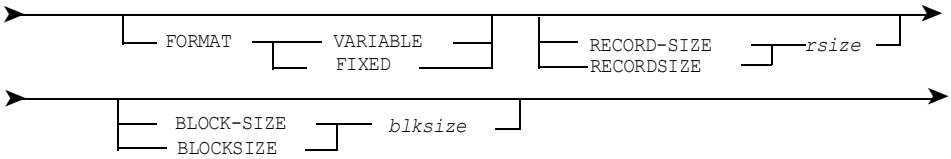
```
 PUBLIC-USER-MEMBER        name   NEW
 PRIVATE-USER-MEMBER              APPEND
 USER-MEMBER                      REPLACE
 SEQUENTIAL file           sequential-options
 PARTITIONED file MEMBER name     NEW
                                  REPLACE
```

where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 CREATE

DB2 CREATE generates an SQL CREATE statement for a DB2 object from its definition in a repository member.

Refer to "DB2 CREATE Syntax 2.5" on page 217 for the syntax of the DB2 CREATE command.

Use the DB2 CREATE command to generate an SQL CREATE statement for a DB2 object defined by the relevant member type in your repository, by entering:

```
DB2 CREATE member ;
```

where *member* is the name of a repository member, of one of these types:

- DB2-ALIAS
- DB2-AUXTABLE
- DB2-DATABASE
- DB2-FUNCTION
- DB2-INDEX
- DB2-PROCEDURE
- DB2-STOGROUP
- DB2-TEMPTABLE
- DB2-TRIGGER
- DB2-TABLE
- DB2-TBSPACE
- DB2-VIEW
- ITEM
- PROGRAM
- DB2-DMS

To generate SQL CREATE SYNONYM statements, use the DB2 SYNONYM command.

The DB2 data types of columns in tables or views is given in the ITEM and GROUP members named in the COLUMNS attribute of the relevant DB2-TABLE or DB2-VIEW member.

If you use distributed databases, you can specify a location as part of an object's name, to uniquely identify it across multiple sites.

Using similar DB2 CREATE commands, you can easily generate similar SQL CREATE statements for objects with different owners. For example, you might want to create copies of a table, for different owners in a project team.

Generated output is displayed on the screen. You can tailor this output, file it on the MP-AID, and/or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits*) while generating output.

The systems administrator can also tailor output using the DB2 profile.

Refer to "Tailoring Output" on page 109 for details of tailoring using the DB2 profile, and user exits.

Refer to "Output Generation Options" on page 334 for details of output generation options.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for an example of a generated SQL CREATE TABLE statement.

Generating SQL COMMENT ON and LABEL ON Statements

Use the WITH-COMMENTS or WITH-LABELS keywords to generate SQL COMMENT ON or LABEL ON and CREATE statements with the DB2 CREATE command. The comments or labels are taken from the DB2-COMMENT or DB2-LABEL attributes of the specified repository member.

To generate SQL COMMENT ON statements, enter:

```
DB2 CREATE member WITH-COMMENTS ;
```

where *member* is the name of a DB2-TABLE, DB2-VIEW, or DB2-ALIAS repository member.

To generate SQL LABEL ON statements, enter:

```
DB2 CREATE member WITH-LABELS ;
```

Alternatively, the systems administrator can change the DB2 profile to automatically generate SQL COMMENT ON or SQL LABEL ON statements when you generate SQL CREATE statements.

Note:

The WITH-COMMENTS/WITH-LABELS keywords override any default DB2 profile settings.

To generate SQL COMMENT ON and LABEL ON statements only, use the DB2 COMMENT and DB2 LABEL commands respectively.

Refer to "Tailoring Output" on page 109 for details of the DB2 profile.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for an example of SQL CREATE, COMMENT ON, and LABEL ON statements generated by the DB2 CREATE command.

Taking User Exits

To take a user exit, enter:

```
DB2 CREATE member USING exit-routine ;
```

where:

member is the name of a DB2-ALIAS, DB2-DATABASE, DB2-INDEX, DB2-STOGROUP, DB2-TABLE, DB2-TBSPACE, or DB2-VIEW member

exit-routine is the name of an executive routine

Note:

The systems administrator can alter the DB2 profile so that a default user exit is always taken when you use the DB2 CREATE command. The USING keyword overrides any default user exits set this way.

Refer to "Output Generation Options" on page 334 for further details of user exits.

Specifying an Owner

To generate an SQL CREATE statement, specifying an owner (overriding any name given in the relevant repository member), enter:

```
DB2 CREATE member SQLID owner ;
```

where:

member is the name of a DB2-TABLE, DB2-VIEW, or DB2-INDEX member

owner is a delimited string of up to 8 characters, giving the ID of a specific user.

Specifying a Location

To generate an SQL CREATE statement, naming a location (overriding any location given for the relevant repository member), enter:

```
DB2 CREATE member LOCATION location ;
```

where:

member is the name of a DB2-TABLE or DB2-VIEW member

location is a delimited string of up to 16 characters, giving a DB2 location.

Refer to "Interrogating Your DB2 Dictionary Schema" on page 102 for details of deriving external names.

Expanding Nested Data Structures

Use the EXPAND keyword to expand the contents of GROUP members (documenting columns) contained in a repository member (documenting a DB2 table, view, or index). These GROUPs can contain other GROUPS, which are expanded in turn.

To expand the contents of a GROUP, enter:

```
DB2 CREATE member EXPAND ;
```

where *member* is the name of a DB2-TABLE, DB2-VIEW, or DB2-INDEX member.

The NO-EXPAND keyword has the opposite effect, forcing all expansion off, overriding any EXPAND attribute in the member definition.

To force expansion off, enter:

```
DB2 CREATE member NO-EXPAND ;
```

Note:

If the EXPAND/NO-EXPAND keyword is not used, expansion only occurs when any EXPAND attributes in the member are specified.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of the EXPAND attribute.

Suppressing Referential Integrity

Use the SUPPRESS-RI keyword to suppress the generation of any referential integrity attributes (constraints, foreign keys) held in a DB2-TABLE member. The table, when created in your DB2 environment using the generated SQL CREATE TABLE statement, will have no referential integrity.

For example, you might use this keyword when you first create a table, to test its basic structure.

To suppress the generation of referential integrity, enter:

```
DB2 CREATE table SUPPRESS-RI ;
```

where *table* is the name of a DB2-TABLE member.

You can define a table with a self-referencing constraint. The constraint is not contained in the SQL CREATE TABLE statement generated, but is automatically generated in a subsequent (additional) ALTER TABLE statement, as required by DB2.

Name Editing Options

You can use the REPLACING/REPLACE, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

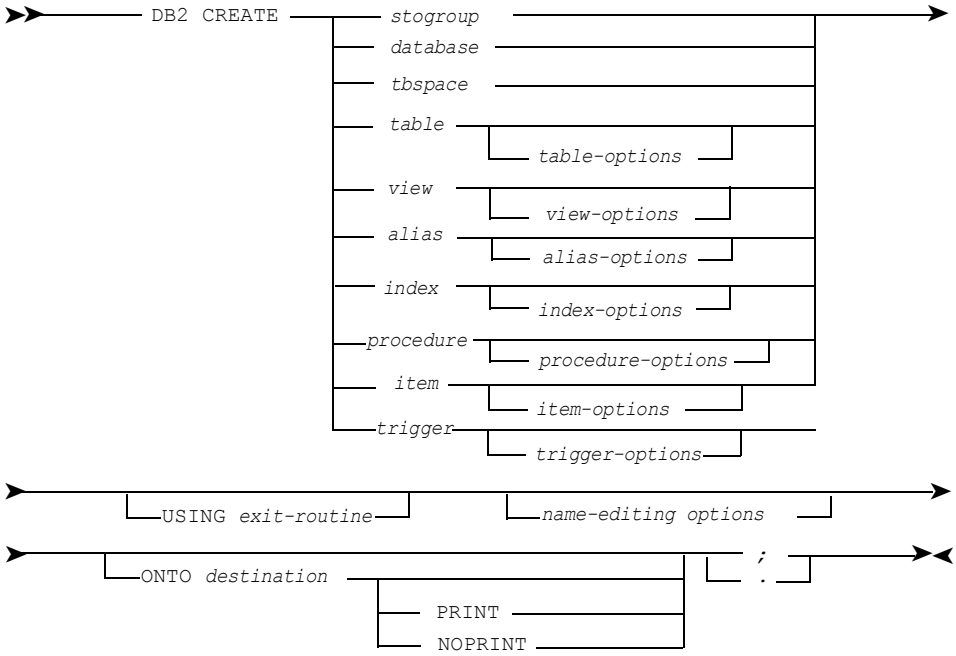
Output Generation Options

Use the ONTO keyword to direct your generated output to:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

Refer to "Output Generation Options" on page 334 for further details of output generation options.

DB2 CREATE Syntax

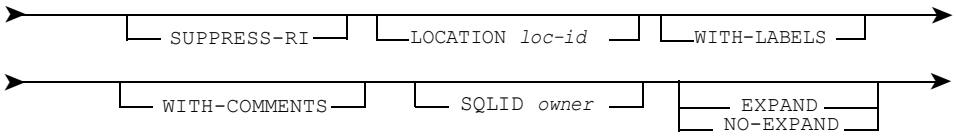


where:

alias, *database*, *index*, *stogroup*, *table*, *tablespace*, *procedure*, *item*, *trigger*, *view*, and *dms* are names of DB2-ALIAS, DB2-DATABASE, DB2-INDEX, DB2-STOGROUP, DB2-TABLE, DB2-TBSPACE, DB2-VIEW, and DB2-DMS repository members.

exit-routine is the name of an executive routine.

table-options are:

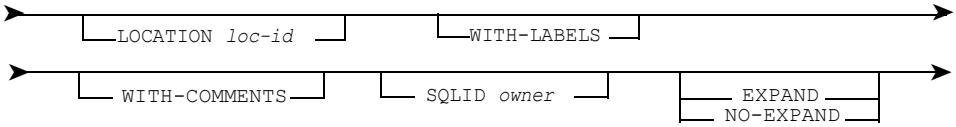


where:

loc-id is a delimited string of up to 16 characters, giving the name of a DB2 location

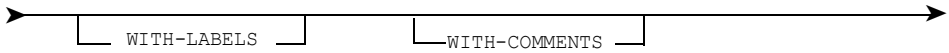
owner is a delimited string of up to 8 characters, giving the ID of a specific user.

view-options are:

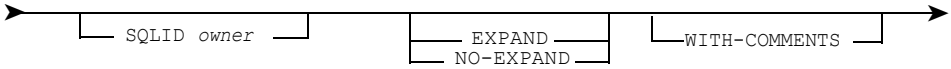


where *loc-id* and *owner* are as defined above.

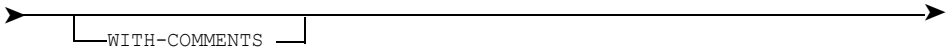
alias-options are:



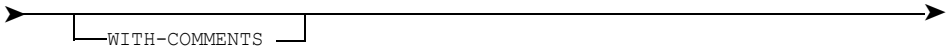
index-options are:



procedure-options:

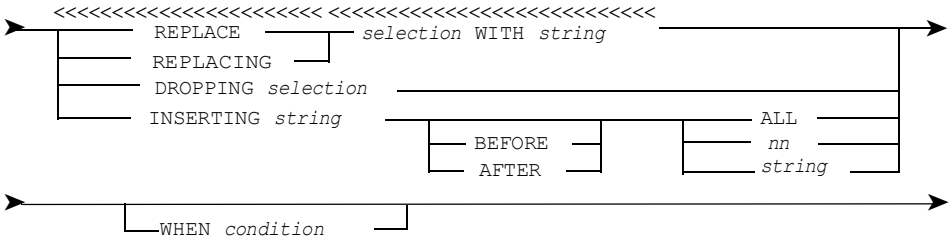


trigger-options:

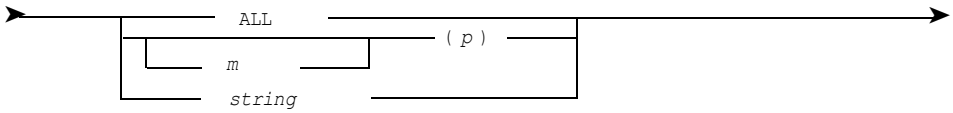


where *owner* is as defined above.

name-editing-options are:



where *selection* is:



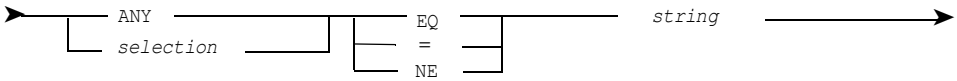
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters.

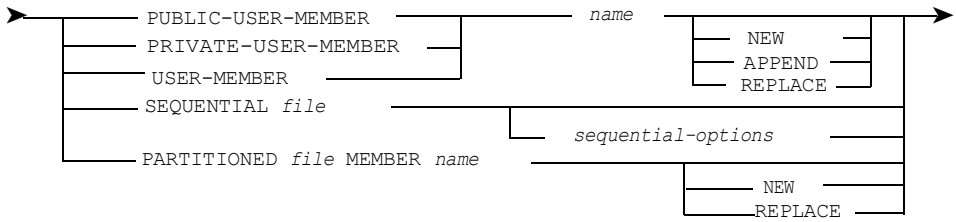
nn is an unsigned integer in the range 1 to 96

condition is:



where *selection* and *string* are as defined above.

destination is:

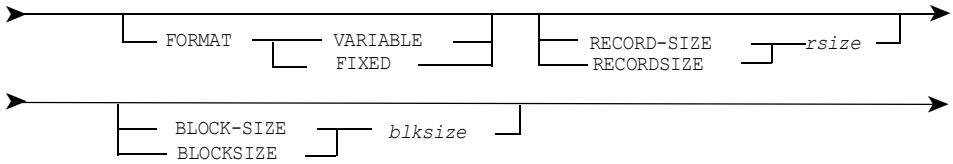


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 DEBUG

DB2 DEBUG produces diagnostic information during DB2 export commands.

Refer to "DB2 DEBUG Syntax" on page 223 for the syntax of the DB2 DEBUG command.

Use the DB2 DEBUG command to give information enabling you to diagnose errors and tailor output. DB2 DEBUG can be set to display a selection of information, including one or more of these:

- Workbench Translation Area (WBTA) command variables
- Contents of the output buffer and associated error messages
- Any user-exits specified.

This information is displayed during execution of an export command.

Any display settings specified are superseded by settings specified when you next use the DB2 DEBUG command.

You can set debugging either on or off. If you set debugging on, you can select one or more types of debug options. If you set debugging off, all selected debug options are turned off.

You can also find out which debug options have been set (if any), and the current display width.

Displaying Information During Export

To show the current debug settings, enter:

```
DB2 DEBUG ;
```

This shows any debug settings specified, and the current display width.

To turn all debugging off, enter:

```
DB2 DEBUG OFF ;
```

To turn one or more debug settings on, enter:

```
DB2 DEBUG ON debug-settings ;
```

where *debug-settings* can be any of these:

- UNVERIFIED-OBJECT
- VERIFIED-OBJECT
- OUTPUT-BUFFER
- EXIT-TABLE
- ENVIRONMENT

To alter the output display width, enter:

```
DB2 DEBUG ON debug-option WIDTH w ;
```

where *w* is the required display width; this defaults to 75.

Obtaining Different Types of Diagnostic Information

Different debug options give different types of information, at different stages of export. You can select sets of appropriate debug options to combine displays.

To display the names and contents of all Workbench Translation Area (WBTA) command variables for a DB2 object, before final SQL name verification, enter:

```
DB2 DEBUG ON UNVERIFIED-OBJECT ;
```

To display WBTA command variables for a DB2 object, after final SQL name verification and before generation of the output, enter:

```
DB2 DEBUG ON VERIFIED-OBJECT ;
```

To display the output buffer (showing both the generated output requested and the corresponding in-context error messages for each line), immediately before it is written to the destination, enter:

```
DB2 DEBUG ON OUTPUT-BUFFER ;
```

To display the current output user-exits, enter:

```
DB2 DEBUG ON EXIT-TABLE ;
```

To display the environmental values stored in command variables, enter:

```
DB2 DEBUG ON ENVIRONMENT ;
```

Output

The output from the DB2 DEBUG command depends on which debug options have previously been set. Groups of tables are always displayed, giving information about command variables for the particular export command being processed.

Note:

These tables bear no relation to tables held on your database: They refer to related sets of command variables.

Command variables have sets of general prefixes, and specific names for each variable. General prefixes follow Manager Products naming conventions. Each general prefix is the heading for a table, and starts with `mpdy_`. Each specific variable is a column in this table.

For example, in a VERIFIED-OBJECT display, for the command variable MPDY_OBJ_BASE_MEMBER_TYPE the general prefix is MPDY_OBJ_ and the specific name is BASE_MEMBER_TYPE. This gives the column BASE_MEMBER_TYPE in the table mpdy_obj_ and one entry in this column might be DB2-DATABASE. This example is demonstrated below:

```
mpdy_obj_.vector
                  | BASE_MEMBER_TYPE |
                  |-------------------|
                  | 'DB2-DATABASE'    |
                  =====================
```

Variable naming conventions are described in more detail below.

UNVERIFIED-OBJECT and VERIFIED-OBJECT display variables have similar names to those of variables generated by the DACCESS, DEXPAND, or DRETRIEVE commands. For example, the variable MPDY_OBJ_MEMBER_TYPE in the VERIFIED-OBJECT display corresponds to the variable MEMBER_TYPE obtained from the DACCESS command. Prefixes in UNVERIFIED- or VERIFIED-OBJECT displays relate to the type of object being exported; MPDY_OBJ_ for any type of object, MPDY_TAB_ for tables, and so on.

In ENVIRONMENT, OUTPUT-BUFFER, and EXIT-TABLE displays, names of specific variables are self-explanatory. For example, the command variable MPDY_ENV_CURR_COMMAND in the ENVIRONMENT display gives the name of the command currently being processed.

OUTPUT-BUFFER displays have two types of variable, prefixed by either MPDY_OUT_ for the output produced, or MPDY_MESS_ for messages associated with that output.

EXIT-TABLE displays are useful when you have some user exits specified; if so, the names and uses of these output-exits are given.

ENVIRONMENT displays give details of the command being processed or of the current session.

Refer to "Tailoring Output" on page 109 for details of user exits.

DB2 DEBUG Syntax

```
>>──── DB2 DEBUG ──┬──────────────────────────────────────────────────────┬──>
                   │ OFF                                                  │
                   │ <<<<<<<  ,  <<<<<<<                                  │
                   └ ON ──┬─ OUTPUT-BUFFER ──────────┬──┬──────────────┬──┘
                          ├─ VERIFIED-OBJECT ────────┤  │ WIDTH w      │
                          ├─ UNVERIFIED-OBJECT───────┤  └──────────────┘
                          ├─ EXIT-TABLE ─────────────┤
                          └─ ENVIRONMENT ────────────┘

>──────┬─ ; ─┬──────────────────────────────────────────────────────────><
       └─ . ─┘
```

where *w* is the display width.

DB2 DECLARE

DB2 DECLARE generates an SQL DECLARE TABLE statement in COBOL, PL1, or Assembler language, from a DB2-TABLE or DB2-VIEW repository member.

Refer to "DB2 DECLARE Syntax" on page 226 for the syntax of the DB2 DECLARE command.

To generate an SQL DECLARE TABLE statement, enter:

```
DB2 DECLARE language FROM member ;
```

where:

language is ASSEMBLER (or ALC, ASM, or BAL), PL1 (or PLI, PL/I, or PL/l) or COBOL.

member is the name of a DB2-TABLE or DB2-VIEW repository member.

The DB2 data type for a column in a table or view is given in the ITEM or GROUP members named in the CONTAINS attribute of the DB2-TABLE or DB2-VIEW member from which the SQL statement is being generated.

If you use distributed databases, you can refer to a table or view using a *location* as part of that object's name, to uniquely identify an object across multiple sites.

Generated output contains any NOT NULL and/or NOT NULL WITH DEFAULT attributes defined in the relevant DB2-TABLE member, or referred to by the relevant DB2-VIEW member.

Generated output is displayed on the screen. You can tailor this output, file it on the MP-AID or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits*) during output. This process is known as taking user exits.

The systems administrator can tailor output by altering the DB2 profile.

Refer to "Tailoring Output" on page 109 for details of the DB2 profile, and user exits.

Refer to "Documenting DB2 Security Information" on page 96 for details of generating column data types.

Taking User Exits

To take a user exit, enter:

```
DB2 DECLARE language FROM member USING exit-routine ;
```

where:

member is the name of a DB2-TABLE or DB2-VIEW repository member.

exit-routine is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken with the DB2 DECLARE command. The USING keyword overrides any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Specifying an Owner

To name an object with a specified owner (overriding any owner specified in the relevant member), enter:

```
DB2 DECLARE language FROM member SQLID owner ;
```

where:

member is a DB2-TABLE or DB2-VIEW member

owner is a delimited string of up to 8 characters, giving the ID of a specific user.

Specifying a Location

To name a location for a DB2-TABLE or DB2-VIEW member, overriding any location defined for that member, enter:

```
DB2 DECLARE language FROM member LOCATION location ;
```

where:

member is a DB2-TABLE or DB2-VIEW member

location is a delimited string of up to 16 characters, giving the object's location.

Refer to "Interrogating Your DB2 Dictionary Schema" on page 102 for details of deriving external names.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of the DB2-USER member type.

Expanding Nested Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for this member, you should use the same keywords for this command, by entering either:

```
DB2 DECLARE language FROM member EXPAND ;
```

or:

```
DB2 DECLARE language FROM member NO-EXPAND ;
```

where *member* is the name of a DB2-TABLE or DB2-VIEW repository member.

EXPAND clauses in the DB2-TABLE member are used as the default.

Refer to "DB2 CREATE" on page 212 for use of the EXPAND/NO-EXPAND keywords.

Refer to Chapter 9, "Repository Member Types," on page 341 for details of the EXPAND attribute.

Output Generation Options

Use the ONTO keyword to direct your generated SQL statements to a specific destination. This destination can be:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

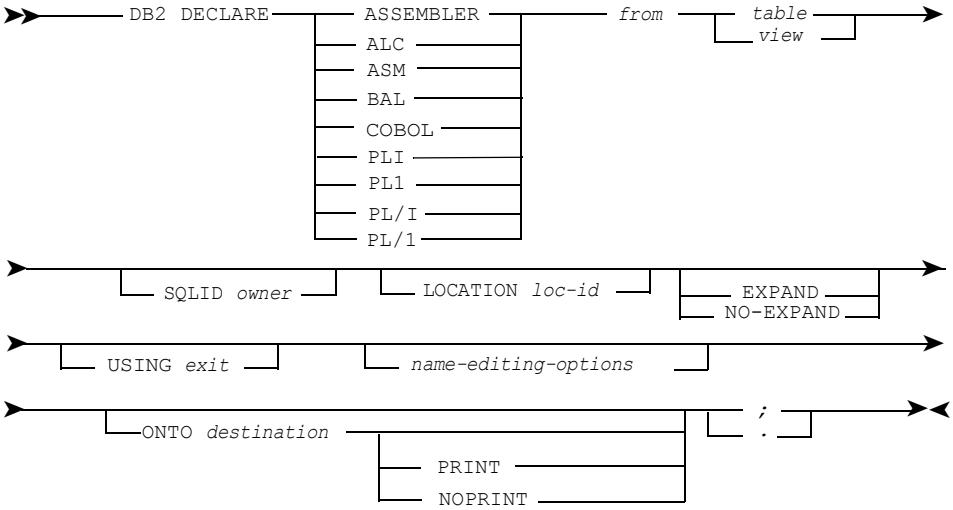
Refer to "Output Generation Options" on page 334 for further details of output generation options.

Name Editing Options

Use the REPLACE/REPLACING, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

DB2 DECLARE Syntax



where:

`table` is the name of a DB2-TABLE repository member.

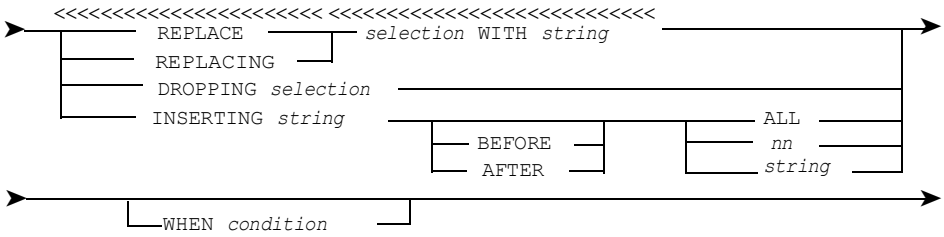
`view` is the name of a DB2-VIEW repository member.

`owner` is a delimited string of up to 8 characters, giving the authorization ID of a particular user.

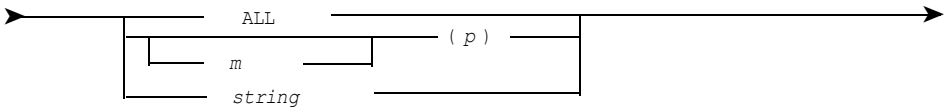
`loc-id` is a delimited string of up to 16 characters, giving the name of a DB2 location.

`exit` is the name of a user-created executive routine.

`name-editing-options` are:



where `selection` is:



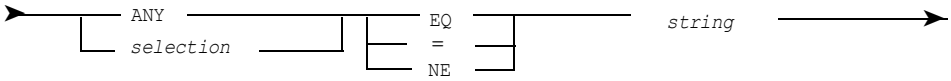
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters.

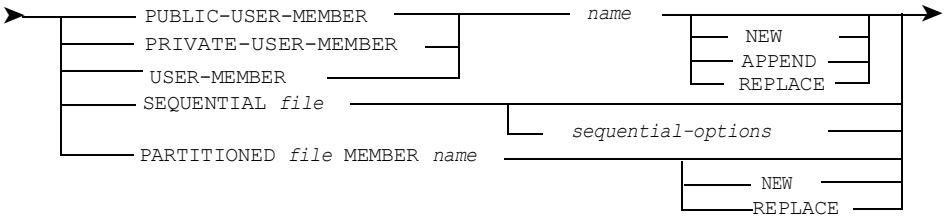
nn is an unsigned integer in the range 1 to 96

condition is:



where *selection* and *string* are as defined above.

destination is:

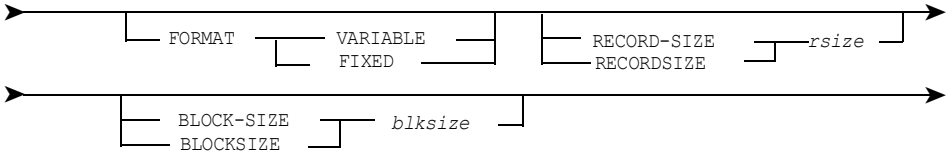


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 DROP

DB2 DROP generates an SQL DROP statement and an optional impact analysis report.

Refer to "DB2 DROP Syntax" on page 232 for the syntax of the DB2 DROP command.

To generate an SQL DROP statement, and an impact analysis report, for a DB2 object from its repository member definition, enter:

```
DB2 DROP member ;
```

where *member* is the name of a DB2-ALIAS, DB2-DATABASE, DB2-INDEX, DB2-STOGROUP, DB2-TABLE, DB2-TBSPACE, DB2-PROCEDURE, DB2-TRIGGER, ITEM, or DB2-VIEW repository member.

To generate SQL DROP SYNONYM statements, use the DB2 SYNONYM command.

If you execute an SQL DROP statement in your DB2 environment, the named DB2 object and all DB2 objects dependent upon it are dropped. For example, if you drop a table, all views and indexes dependent on that table are also dropped. So the DB2 DROP command also generates an impact analysis report, showing the impact of dropping the specified member in your DB2 environment.

You can generate SQL DROP statements which DB2 may reject (such as an SQL statement to drop a storage group containing table spaces), as the impact analysis report generated can be useful (for example, if a disk-pack holding the storage group became unavailable, you would want to know what is impacted).

The generated output is displayed on the screen. You can tailor the SQL DROP statements, file them on the MP-AID, or send them to an external dataset, using output generation options. You can also tailor the SQL statements by calling executive routines (*user exit routines*) at set points (*user exits)* during output. This process is known as taking user exits. You cannot alter the impact analysis report using these methods; use the REPORT keyword for this purpose.

The systems administrator can tailor both the SQL statements and the impact analysis report by altering the DB2 profile.

For further details of tailoring, refer to "Tailoring Output" on page 109.

The Impact Analysis Report

By default, an impact analysis report is generated with the DB2 DROP command, showing a hierarchy of the impacted repository members (and so the objects in the DB2 environment) that would be dropped or affected if that SQL DROP statement is applied to your DB2 environment. The first line in the report gives the member to be dropped. Each subsequent line shows an impacted member.

Each line is numbered, in case of duplication. Entries that appear twice are only given in full once; the next appearances refer to the line number of the first appearance. For each line, the following is also shown, from left to right:

- The relationship between the impacted member and the member one level of indent above it
- The name of the member
- The owner for that member (if an owner exists)
- The type of member

Synonyms of members are also reported.

An impact analysis report is not generated when a DB2 DROP command is applied to a DB2-INDEX member, as only the specified index would be dropped or affected by the SQL DROP INDEX statement.

To deliberately suppress the generation of the impact analysis report, enter:

```
DB2 DROP member NO-IMPACT-REPORT ;
```

Members impacted by the DB2 DROP command are reported but are not removed from the repository. If you drop the object from the DB2 environment, you should update the repository to reflect the changes, unless you intend to later re-create the object and those dependent on it. The impact analysis report will help you carry out these updates.

Tailoring the Impact Analysis Report

The impact analysis report is tailorable, with a set of rules defining how the repository is searched. The systems administrator can tailor the table which drives the search algorithm to allow extra search paths to be taken which may be used by the user in addition to the standard relationships.

Refer to Chapter 5, "Export to DB2," on page 105 for details of tailoring.

You can also use specific keywords with the DB2 DROP command to alter the report width (overriding the default of 80 characters) and the size of each indent in a nested structure (overriding the default of 5 characters). To specify the width of the report, enter:

```
DB2 DROP member REPORT WIDTH w ;
```

where *w* is the new report width. This must be between 50 and 246 characters.

To specify the size of each indent in a nested structure, enter:

```
DB2 DROP member REPORT INDENT i ;
```

where *i* is the width of each new indent. This must be between 5 and 20 characters.

You can use KEPT-DATA lists to help manage the impacted members. To store all members displayed in the report in an unnamed KEPT-DATA list, enter:

```
DB2 DROP member REPORT KEEP ;
```

To store all reported members in a named KEPT-DATA list, enter:

```
DB2 DROP member REPORT KEEP IN kept-name ;
```

where *kept-name* is the name of a KEPT-DATA list.

To add members to an existing KEPT-DATA list, enter either:

```
DB2 DROP member REPORT ALSO ;
```

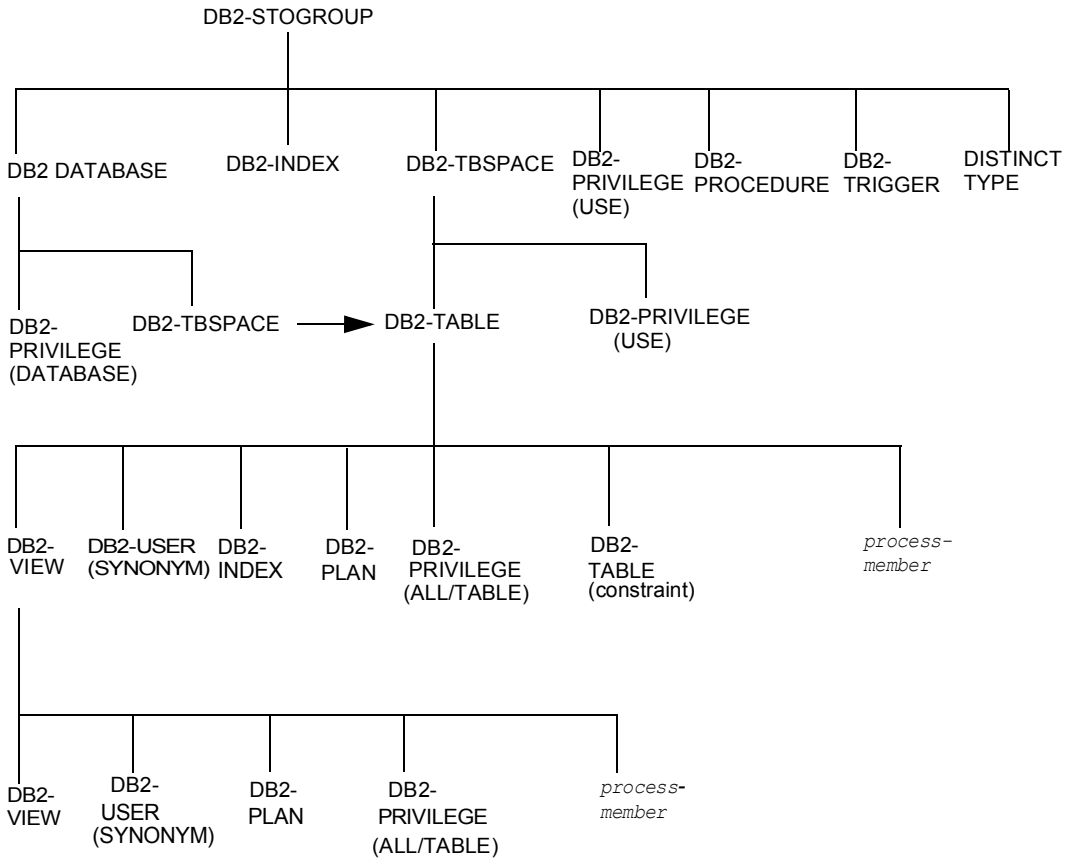
or

```
DB2 DROP member REPORT ALSO IN kept-name ;
```

Example of the Structure of the Impact Analysis Report

The impact analysis report has the structure as seen in Figure 31:

Figure 31 • Impact Analysis Report Structure



where `process-member` is a PROGRAM, MODULE, SYSTEM, or MMR-SYSTEM member.

For example, if a DB2-TABLE member is specified in a DB2 DROP command then an SQL DROP TABLE statement is generated, which if applied to your DB2 environment will drop that table and any views and indexes dependent on it. Synonyms for and privileges on the dropped table and views will also be dropped. SQL statements which refer to these tables and views will be affected. SQL statements can be imbedded in programs themselves contained in plans.

Taking User Exits

To take a user exit, enter:

```
DB2 DROP member USING exit-routine ;
```

where:

member is the name of a DB2-DATABASE, DB2-INDEX, DB2-STOGROUP, DB2-TABLE, DB2-TBSPACE, or DB2-VIEW repository member

exit-routine is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken when you use the DB2 DROP command. The USING keyword overrides any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

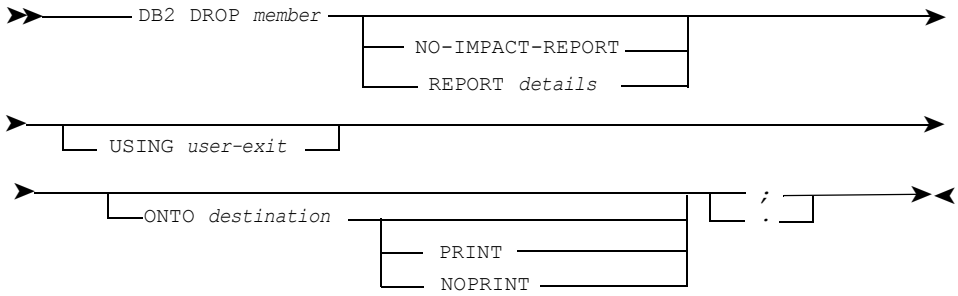
Output Generation Options

Use the ONTO keyword to direct your generated output to:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

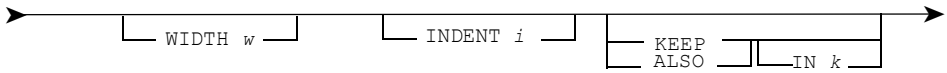
Refer to "Output Generation Options" on page 334 for further details of output generation options.

DB2 DROP Syntax



where *member* is the name of a DB2-ALIAS, DB2-DATABASE, DB2-INDEX, DB2-STOGROUP, DB2-TABLE, DB2-TBSPACE, DB2-PROCEDURE, DB2-TRIGGER, ITEM, or DB2-VIEW repository member.

details are:



where:

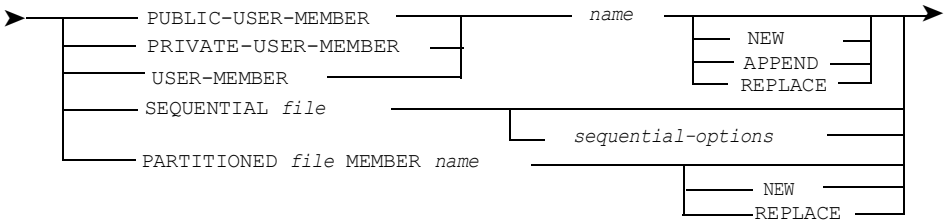
w is an integer between 50 and 246

i is an integer between 5 and 20

k is the name of a KEPT-DATA list.

where *user-exit* is the name of an executive routine.

destination is:

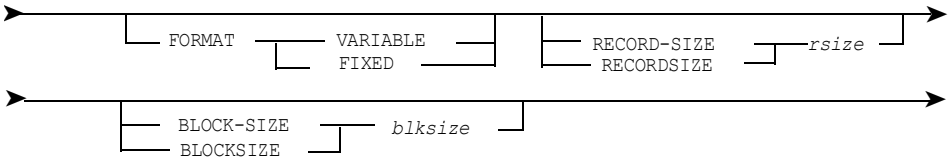


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 GRANT and DB2 REVOKE

DB2 GRANT and DB2 REVOKE generates an SQL GRANT or REVOKE statement for a DB2 object from its definition in a DB2-PRIVILEGE repository member.

Refer to "DB2 GRANT and DB2 REVOKE Syntax" on page 237 for the syntax of the DB2 GRANT command.

Use the DB2 GRANT or DB2 REVOKE commands to generate respectively SQL GRANT or REVOKE statements, by entering either:

```
DB2 GRANT member ;
```

or

```
DB2 REVOKE member ;
```

where `member` is the name of a DB2-PRIVILEGE member.

The SQL GRANT statement will include the WITH GRANT OPTION keyword if the WITH-GRANT-OPTION attribute is present in the DB2-PRIVILEGE member.

You can generate SQL statements to change privileges for all DB2-USER members referred to directly or indirectly by the DB2-PRIVILEGE member. You can also generate SQL statements to change privileges for DB2-USER members representing single-user IDs only.

Generated output displays on the screen. You can tailor this output, file it on the MP-AID, or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits)* during output. This process is known as taking user exits.

The systems administrator can tailor output by altering the DB2 profile.

Refer to "Tailoring Output" on page 109 for details of the DB2 profile and user exits.

Specifying an Owner of an Object

To generate SQL statements to change privileges for a table or view, and override the current owner name, enter either:

```
DB2 GRANT member SQLID owner ;
```

or

```
DB2 REVOKE member SQLID owner ;
```

where:

member is the name of a DB2-PRIVILEGE repository member

owner is a delimited string of up to 8 characters, giving the ID of a specific user.

Expanding Nested Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for a DB2-PRIVILEGE member, you should use the same keywords for this command, by entering:

```
DB2 GRANT member EXPAND ;
```

or

```
DB2 GRANT member NO-EXPAND ;
```

or

```
DB2 REVOKE member EXPAND ;
```

or

```
DB2 REVOKE member NO-EXPAND ;
```

where *member* is the name of a DB2-PRIVILEGE repository member.

EXPAND attributes in the DB2-TABLE member are used as the default.

Refer to "DB2 CREATE" on page 212 for use of the EXPAND/NO-EXPAND keywords.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for further details about the EXPAND attribute.

Changing Privileges on an Expanded Range of Users

To generate SQL statements to change privileges for all DB2-USER members referred to directly or indirectly by a DB2-PRIVILEGE member, enter:

```
DB2 GRANT member USER-EXPANSION FULL ;
```

or

```
DB2 REVOKE member USER-EXPANSION FULL ;
```

where *member* is the name of a DB2-PRIVILEGE member.

Alternatively, to generate SQL statements to change privileges for DB2-USER members representing single-user IDs only, enter either:

```
DB2 GRANT member USER-EXPANSION SINGLE-IDS ;
```

or

```
DB2 REVOKE member USER-EXPANSION SINGLE-IDS ;
```

By default, only privileges on DB2-USER members referred to directly from the DB2-PRIVILEGE member are changed.

Specifying the Grantor of a Privilege

Use the BY-GRANTOR keyword to generate a BY keyword in an SQL REVOKE statement, by entering:

```
DB2 REVOKE member BY-GRANTOR ;
```

where *member* is the name of a DB2-PRIVILEGE repository member.

Taking User Exits

To take a user exit, enter either:

```
DB2 GRANT member USING exit-routine ;
```

or

```
DB2 REVOKE member USING exit-routine ;
```

where:

member is the name of a DB2-PRIVILEGE member

exit-routine is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken when you use the DB2 GRANT or DB2 REVOKE commands. The USING keyword overrides any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Name Editing Options

You can use the REPLACING/REPLACE, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

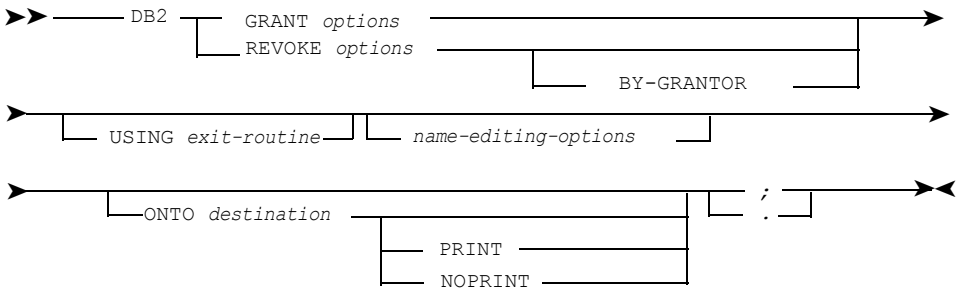
Output Generation Options

Use the ONTO keyword to direct your generated output to:

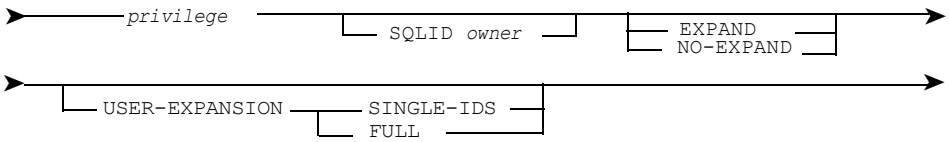
- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

Refer to "Output Generation Options" on page 334 for further details of output generation options.

DB2 GRANT and DB2 REVOKE Syntax



where *options* are:



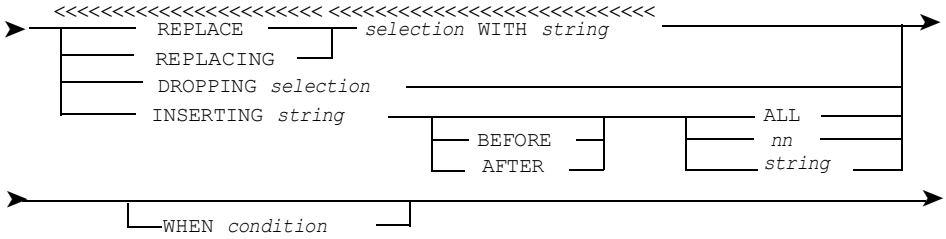
where:

privilege is the name of a DB2-PRIVILEGE member

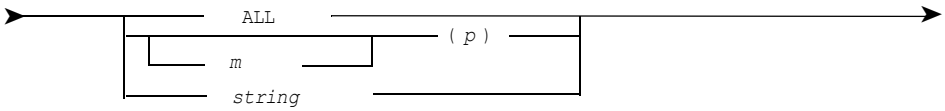
owner is a delimited string of up to 8 characters, giving the authorization ID of a particular user.

exit-routine is the name of an executive routine.

name-editing-options are:



where *selection* is:



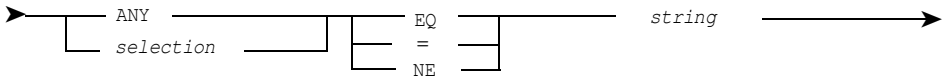
where:

m and *p* are integers in the range 1 to 96

string is a delimited string of not more than 32 printable characters

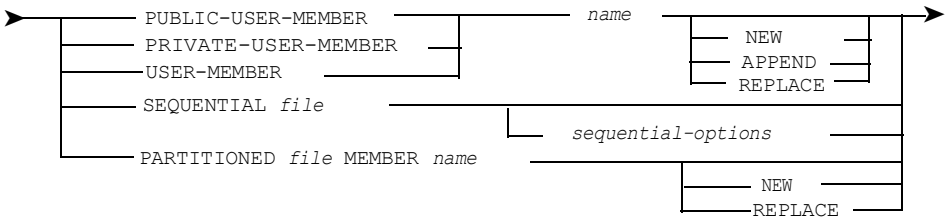
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

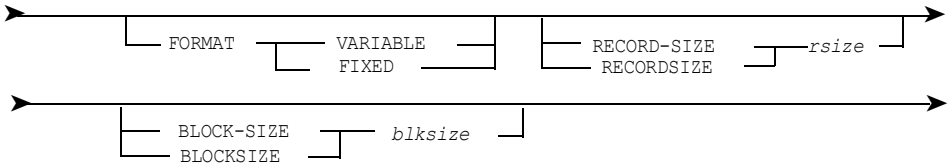


where:

name is the name of a USER-MEMBER

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

DB2 LIST CYCLES

DB2 LIST CYCLES identifies the cycles found in the DB2 design present in the Workbench Design Area (WBDA) and to list the tables which appear within each cycle.

A *cycle* is a path of relationships connecting a table to itself, where the arrows representing the relationships all flow in the same direction. The tables appearing in this path are said to be in *cyclic order*.

To list the tables in each cycle in cyclic order, beginning with the table (in the cycle) having the lowest WBDA number, enter:

```
DB2 LIST CYCLES ;
```

To list the tables in each cycle alphanumerically, enter:

```
DB2 LIST CYCLES ALPHABETICALLY ;
```

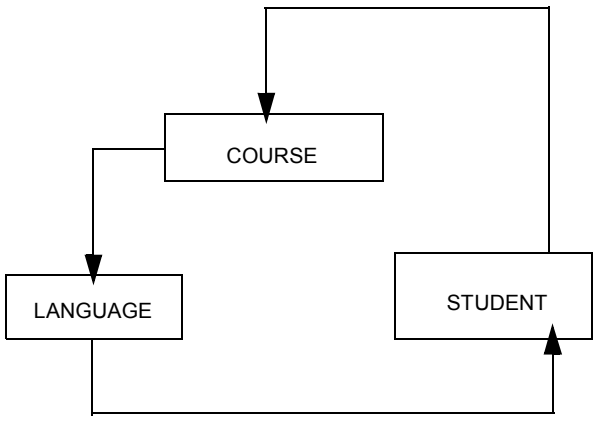
Named tables in the cycle are listed in alphanumerical order of table name, followed by any unnamed tables in order of WBDA number.

The DB2 LIST CYCLES command can be executed only if the WBDA contains normalized data. Otherwise, the command is terminated and a message of explanation is output. If the WBDA contains normalized data but no DB2 design, the command causes the DB2 design to be generated before producing the list.

For each DB2 table appearing in a cycle, the list includes its WBDA number, its primary key, its name (if one has been assigned) and, if the table appears in more than one cycle, the keyword MULTIPLE.

In Figure 32, an example is pictured of a cycle with its path of tables and connecting relationships:

Figure 32 • A Cycle with its Path of Tables and Connecting Relationships



Refer to "Output from the DB2 LIST CYCLES Command" on page 83 for details of the DB2 LIST CYCLES command output

Refer to "Introduction to Referential Structures and Cycles" on page 31 for a further discussion of cycles and how they can affect design decisions.

DB2 LIST CYCLES Syntax

```
>>──DB2 LIST CYCLES──┬──────────────────┬──┬──;──┬──>>
                     └─ALPHABETICALLY───┘  └──.──┘
```

DB2 LIST TABLES

DB2 LIST TABLES produces a list of all or some of the tables appearing in the DB2 design generated in the Workbench Design Area (WBDA).

To list all the tables in the DB2 design in order of WBDA number, enter:

```
DB2 LIST TABLES ;
```

To list all tables alphanumerically, enter:

```
DB2 LIST TABLES ALPHABETICALLY ;
```

Named tables are listed in alphanumeric order of table name, followed by any unnamed tables in order of WBDA number.

To list some of the tables in the DB2 design, you make your selection based on table type. You can select any number of table types in the command.

To list a selection of tables in order of WBDA number, enter:

```
DB2 LIST TABLES selection ;
```

where *selection* is one or more of these keywords:

- ROOTS indicates that every root parent table is to be listed
- PARENTS is used to select every parent table for listing whether it is a root parent or a table which is both a parent and a dependent
- LEAFS or LEAVES is used to select every leaf dependent table for listing
- DEPENDENTS indicates that every dependent table is to be listed whether it is a leaf dependent or a table which is both a dependent and a parent
- INDEPENDENT indicates that every table is to be listed which is neither a parent nor a dependent table, that is, a table which does not participate in any foreign key relationships.

To list all selected named tables alphanumerically followed by any selected unnamed tables in order of WBDA number, enter:

```
DB2 LIST TABLES ALPHABETICALLY selection ;
```

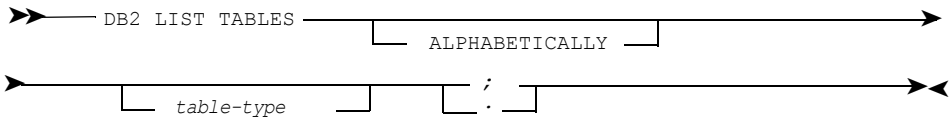
where *selection* is defined as above.

The command can be executed only if the WBDA contains normalized data. Otherwise, the command is terminated and a message of explanation is output. If the WBDA contains normalized data but no DB2 design, the command causes the DB2 design to be generated before producing the list.

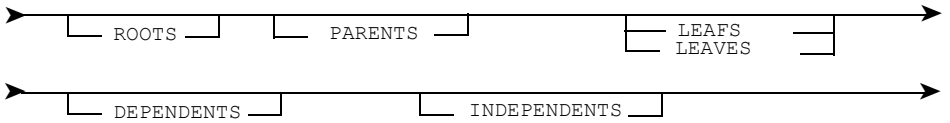
For each DB2 table selected, the list includes the WBDA number of the table, its primary key, its name (if one has been assigned), and its type.

Refer to "Output from the DB2 LIST TABLES Command" on page 81 for details of the DB2 LIST TABLES command output.

DB2 LIST TABLES Syntax



where *table-type* is:



DB2 PLOT CLUSTER

DB2 PLOT CLUSTER produces a DB2 Cluster Plot of all or some of the tables in the DB2 design.

Refer to "DB2 PLOT CLUSTER Syntax" on page 244 for the syntax of the DB2 PLOT CLUSTER command.

Use the DB2 PLOT CLUSTER command to produce a DB2 Cluster Plot of all or some of the tables in the DB2 design generated in the Workbench Design Area (WBDA).

You must enter one (and only one) of the following keywords or clauses in the command to indicate your selection of the tables to be displayed:

- The ALL keyword to select all the tables in the WBDA
- The NAME clause for a selection of tables by name
- The NUMBERS clause for a selection of tables by number.

If you also enter the keyword ALPHABETICALLY, the selected tables will be output alphanumerically.

For each selected table, the output shows a diagram in cluster form of its foreign key relationships, if any, with the other tables in the DB2 design. When all the clusters have been displayed, the DB2 Design Relationship Matrix is output. This is a two-dimensional table which summarizes all of the relationships holding between the tables of the DB2 design, whether or not they have been selected for display.

The command can be used only if the WBDA contains normalized data. If there is no data in the WBDA, or if it has not been normalized, you are informed and the command is terminated. If the WBDA contains normalized data but no DB2 design, this command causes the DB2 design to be generated and then produces the plot.

Refer to "Output from the DB2 PLOT CLUSTER Command" on page 63 for further details of the output of the DB2 PLOT CLUSTER command.

The USING FORMAT option of this command is available only if you have the User Formatted Output facility installed. It allows you to specify a valid FORMAT member of the dictionary in order to tailor the format in which the tables are output.

Displaying All the Tables in the Workbench Design Area

To produce a DB2 Cluster Plot displaying every table in the Workbench Design Area (WBDA), enter:

```
DB2 PLOT CLUSTER ALL ;
```

This displays the tables in order of WBDA number. To display all the tables alphanumerically, enter:

```
DB2 PLOT CLUSTER ALL ALPHABETICALLY ;
```

This causes the named tables to be displayed in alphanumeric order of table name, followed by any unnamed tables in ascending order of WBDA number.

Displaying Tables Selected by Name

To produce a DB2 Cluster Plot displaying tables selected by name, enter:

```
DB2 PLOT CLUSTER NAMES name-list ;
```

where *name-list* is a list of one or more valid names of tables present in the Workbench Design Area (WBDA). Table names in name-list must be separated by commas.

Tables display in the order listed unless the keyword ALPHABETICALLY is specified in the command.

To display the tables in alphanumeric order of table name, enter:

```
DB2 PLOT CLUSTER NAMES name-list ALPHABETICALLY ;
```

For example:

```
DB2 PLOT CLUSTER NAMES DEPARTMENT,OFFICE,EMPLOYEE ALPHABETICALLY ;
```

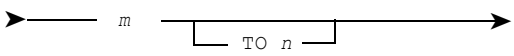
Displaying Tables Selected by Number

This is the only way to select tables in the Workbench Design Area (WBDA) which have not yet been named.

To produce a DB2 Cluster Plot of tables selected by their WBDA number, enter:

```
DB2 PLOT CLUSTER NUMBERS range-list ;
```

where *range-list* is a list of one or more numeric ranges, separated by commas, each of the form:



where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*. Every table is selected whose WBDA number appears in the list or falls within a range appearing in the list. Tables are displayed in the order listed unless the keyword ALPHABETICALLY is also specified in the command.

To display the listed tables alphanumerically, enter:

```
DB2 PLOT CLUSTER NUMBERS range-list ALPHABETICALLY ;
```

This causes the named tables in *range-list* to be displayed in alphanumeric order of table name, followed by any unnamed tables in ascending order of WBDA number. An example of this option is shown below:

```
DB2 PLOT CLUSTER NUMBERS 1,4,6 TO 12,17 TO 20,25 ALPHABETICALLY ;
```

Displaying Tables in a Specific Format

To produce a cluster plot of tables in a format tailored to your requirements, enter:

```
DB2 PLOT CLUSTER selection USING FORMAT format-member ;
```

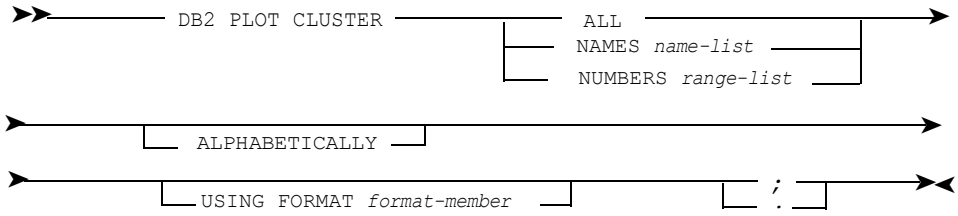
where:

format-member is the name of a previously defined FORMAT repository member. Tables are output according to the specifications in the FORMAT member.

selection is one of the following:

- ALL
- NAMES *name-list*
- NUMBERS *range-list*

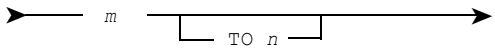
DB2 PLOT CLUSTER Syntax



where:

name-list is a list of validly named tables in the WBDA. If there are two or more names in the list they must be separated by commas

range-list is a list of one or more numeric ranges, separated by commas, each of the form:



where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*.

format-member is the name of a previously defined, valid FORMAT member.

DB2 PLOT REFERENTIAL-STRUCTURES

DB2 PLOT produces a DB2 Referential Structures Plot of one or all of the referential structures in the DB2 design.

Refer to "DB2 PLOT Syntax" on page 250 for the syntax of the DB2 PLOT command.

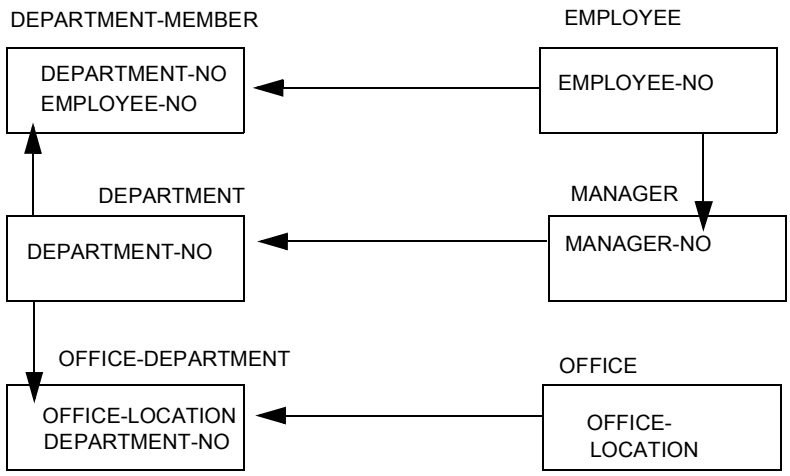
Use the DB2 PLOT REFERENTIAL-STRUCTURES command to produce the DB2 Referential Structures Plot, a consolidated overview display of one or all of the referential structures in the DB2 design present in the Workbench Design Area (WBDA).

The command can be executed only if the WBDA contains normalized data. Otherwise, the command is terminated and a message of explanation is output. If the WBDA contains normalized data but no DB2 design, the command causes the DB2 design to be generated before producing the list.

A referential structure can be described as a set of tables and relationships such that each table in the set is either a parent or a dependent of itself or of some other table in the set. Every table that is a parent or dependent in the set is part of exactly one referential structure.

Figure 33 shows a referential structure in the case of the Department Model example:

Figure 33 • Referential Structure in the Case of the Department Model



Refer to "Department Model Example" on page 37 for details of the Department Model example.

For each referential structure displayed in a DB2 Referential Structures Plot, one or more individual hierarchical plots are produced, each starting with a seed table. The seed used in the (first) plot for the first referential structure displayed is called the *primary seed*. Any other plots required for any of the structures displayed are called *additional plots* beginning with *additional seeds*.

Refer to "Layout" on page 72 for a full description of the Referential Structures Plot and its layout.

You can display all of the referential structures of the DB2 design present in the WBDA by specifying the ALL keyword in the command, or you can indicate that only a single referential structure is to be displayed by including a SEED clause specification. Either ALL or SEED must be specified.

You can further specify:

- The keyword PARENTS to indicate that only parent tables and relationships are to be displayed in the plot, or
- The keyword DEPENDENTS to indicate that only dependent tables and relationships are to be displayed.

If both parent and dependent tables and relationships are to be displayed, then neither PARENTS nor DEPENDENTS should be specified.

Refer to "Introduction to Referential Structures and Cycles" on page 31 for a further discussion of referential structures.

Refer to "Output from the DB2 PLOT REFERENTIAL-STRUCTURES Command" on page 71 for further details of the DB2 PLOT REFERENTIAL-STRUCTURES command output.

Displaying All Referential Structures

Use the ALL keyword in the DB2 PLOT REFERENTIAL-STRUCTURES command to display all of the referential structures of the DB2 design in the DB2 Referential Structures Plot. For each structure displayed, this will cause one or more hierarchical plots to be produced representing every table and relationship in the structure. Each independent table, if any, is also displayed in an additional seed-only plot. Specifying the ALL keyword is the only way to ensure that every table and relationship in the DB2 design is displayed in the DB2 Referential Structures Plot.

How the tables and relationships are displayed and whether or not each referential structure can be displayed in a single hierarchical plot, depends on whether one or the other (or neither) of the PARENTS and DEPENDENTS keywords is also specified in the command.

To depict each referential structure by a single hierarchical plot, enter:

```
DB2 PLOT REFERENTIAL-STRUCTURES ALL ;
```

without specifying either PARENTS or DEPENDENTS.

Then, for each referential structure in the DB2 design, beginning with the seed, the plot includes the entire referential structure, displaying all the remaining tables in the structure, both dependent and parent, and all the foreign key relationships.

The seed for each plot is selected automatically, as follows:

- If there are any root parent tables in the DB2 design, the root parent whose number in the Workbench Design Area (WBDA) is the lowest is chosen as the primary seed
- Each additional seed, in turn, is the lowest numbered root parent table which has not already been displayed
- If, at any point, there are remaining referential structures in the design (and, therefore, additional plots required), but no remaining root parent tables, that is, each remaining structure contains one ore more cycles instead of root parents, then ASG-DesignManager selects as the next seed the lowest numbered (non-independent) table remaining in the DB2 design. This is repeated until all the tables of all the referential structures have been displayed.
- Finally, if there are any independent tables in the DB2 design, each is displayed as a seed-only additional plot. They are selected for display in order of WBDA number.

To display only dependent tables and relationships (following the seed) in each hierarchical plot, enter:

```
DB2 PLOT REFERENTIAL-STRUCTURES ALL DEPENDENTS ;
```

This does not ensure that each referential structure can be displayed in a single hierarchical plot. Additional plots may be required to complete the display.

Seeds for the plots are selected automatically in the same way as selected when neither DEPENDENTS nor PARENTS is specified (beginning with the lowest numbered root parent table in the DB2 design, as indicated above). Thus, a separate hierarchical plot is produced for each root parent table that has not already been displayed. Although more than one plot may belong to the same referential structure, the display produced by this variant of the command is often in the most convenient form for the user.

To display only parent tables and relationships (following the seed) in each hierarchical plot, enter:

```
DB2 PLOT REFERENTIAL-STRUCTURES ALL PARENTS ;
```

Then, as with DEPENDENTS, each referential structure may not be depicted by a single hierarchical plot. Additional plots may be required.

The seed for each plot is selected automatically, as follows:

- If there are any leaf dependent tables in the DB2 design, the leaf dependent with the lowest WBDA number is chosen as the primary seed
- Each additional seed, in turn, is the lowest numbered leaf dependent table which has not already been displayed
- If, at any point, there are remaining referential structures in the design (and, therefore, additional plots required), but no remaining leaf dependent tables, that is, each remaining structure contains one or more cycles instead of leaf dependents, then ASG-DesignManager selects as the next seed the lowest numbered (non-independent) table remaining in the DB2 design. This is repeated until all the tables of all the referential structures have been displayed.
- Finally, if there are any independent tables in the DB2 design, each is displayed as a seed-only additional plot They are selected for display in order of WBDA number.

Thus, a separate hierarchical plot is produced for each leaf dependent table that has not already been displayed.

Displaying a Single Referential Structure

Use the SEED clause, with a table specified as seed, in the DB2 PLOT REFERENTIAL-STRUCTURES command to display all or part of a single referential structure (or a single independent table) from the DB2 design present in the Workbench Design Area (WBDA). Just one hierarchical plot is produced.

Starting with the specified seed, the plot displays a related set of tables and their connecting relationships from the referential structure in which the selected seed appears. (Recall that a table that participates in a relationship appears in one and only one referential structure.) How the tables and relationships are displayed and whether or not the entire referential structure appears in the plot depends on whether one or the other (or neither) of the PARENTS and DEPENDENTS keywords has also been specified in the command and also on whether the seed table specified is a parent or dependent (or independent) table and whether or not it is a root or a leaf.

You can use the DB2 LIST TABLES command to help you choose appropriate seeds, because the output produced indicates the table type; that is, it identifies root, leaf, parent, dependent, and independent tables.

To ensure that the hierarchical plot produced represents the entire referential structure in which the seed appears, enter:

```
DB2 PLOT REFERENTIAL-STRUCTURES SEED selection ;
```

without specifying either PARENTS or DEPENDENTS.

where *selection* is one of the following:

- NUMBER *number*, or
- NAME *name*.

  where *number* or *name* identifies, by WBDA number or table name, respectively, a non-independent table of the DB2 design.

Then, beginning with the specified seed, the plot includes all of the remaining tables in the structure, both dependent and parent, and all of the foreign key relationships. (If, on the other hand, an independent table is specified in the SEED clause, then only the seed table will be displayed in the plot.)

To display only dependent tables and relationships (following the seed) in the plot, enter:

```
DB2 PLOT REFERENTIAL-SIRUCTURES SEED selection DEPENDENTS ;
```

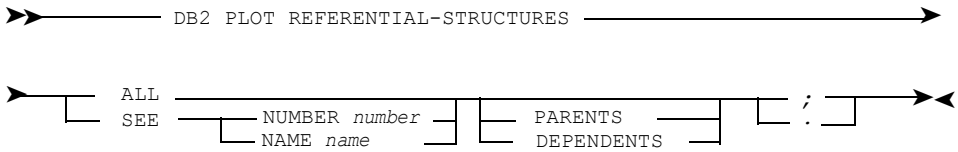
No parent relationships are traversed and, therefore, the only tables displayed following the seed are its descendants in the referential structure. As a consequence, the entire referential structure in which the seed appears may not be represented in the plot. (If the specified seed is a leaf table or an independent table, then only the seed table will be displayed.)

To display only parent tables and relationships (following the seed), enter:

```
DB2 PLOT REFERENTIAL-STRUCTURES SEED selection PARENTS ;
```

No dependent relationships are traversed; therefore, no descendants of the seed are displayed. As a consequence, the entire referential structure in which the seed appears may not be represented in the plot. (If the specified seed is a root table or an independent table, then only the seed table will be displayed.)

DB2 PLOT Syntax



where:

number is a table number in the Workbench Design Area

name is a table name.

DB2 POPULATE

DB2 POPULATE populates the repository with DB2-TABLE, DB2-INDEX, and DB2-VIEW members, generated from the DB2 design, and to produce a report of the generated members.

Refer to "DB2 POPULATE Syntax" on page 260 for the syntax of the DB2 POPULATE command.

The DB2 POPULATE command generates dictionary member definitions and populates the dictionary with them. A report of the generated member definitions is automatically output.

The DB2 POPULATE command generates dictionary member definitions for one or more of the following member types, from selected tables of the DB2 design in the Workbench Design Area (WBDA):

- DB2-TABLE
- DB2-INDEX
- DB2-VIEW

Each time you issue the command, a SYSTEM member also can be generated and placed in the dictionary, containing a list of all the DB2 dictionary members generated by the command.

The command can be used only if the WBDA contains normalized data. If there is no data in the WBDA, or if the data has not been normalized, the command is terminated and a message to that effect is output.

The DB2 POPULATE command will automatically generate the DB2 design if one has not already been generated. Any unnamed tables will be ignored by the command.

Although the content of a generated DB2 dictionary member definition is only a subset of the permissible content (which can be added later by a user if required), the generated definitions are complete enough to be used subsequently to produce valid SQL CREATE TABLE, CREATE INDEX, and CREATE VIEW statements. (When creating the DB2 object, DB2 will assign default values to all the remaining clauses.)

Each member definition is preceded by an ADD command and followed by a terminator.

DB2 POPULATE automatically generates primary key keywords and foreign key clauses in DB2-TABLE definitions to support referential integrity, unless you use the NO-RI option to suppress them. In addition, the command enables you to assign DB2 tables to specific tablespaces (the TBSPACE option).

The command also allows you to associate a dictionary DB2-USER member (via the CREATOR-OWNER clause) with the DB2 dictionary members being defined.

You must enter one (and only one) of the following keywords or clauses in the command to indicate your selection of the DB2 tables in the WBDA to be used in generating the dictionary definitions:

- The ALL keyword to select all the tables in the WBDA
- The NAMES clause for a selection of tables by name
- The NUMBERS clause for a selection of tables by WBDA number.

If you also enter the keyword ALPHABETICALLY, the definitions are generated (and displayed) in alphanumeric order of table name.

The USING FORMAT option of this command is available only if you have the User Formatted Output facility installed. It allows you to specify a defined FORMAT member of the dictionary to control the format in which the member definitions are generated.

You can generate dictionary definitions for any or all of the member types in one command, but you must specify them in the order in which the corresponding clauses appear in the command syntax.

By prefixing DB2 POPULATE with a NOPRINT command you can stop any output being printed.

Refer to *ASG-ControlManager User's Guide* for details of the NOPRINT command.

Generating and Populating DB2-TABLE Members

Use the keyword TABLES to generate and populate the dictionary with DB2-TABLE members, one for each selected table in the Workbench Design Area (WBDA). The name of the table in the WBDA becomes the name of the generated DB2-TABLE dictionary member.

To populate the dictionary with DB2-TABLE members, enter:

```
DB2 POPULATE TABLES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The command populates the dictionary with DB2-TABLE members which automatically contain primary key keywords and foreign key clauses to support referential integrity (RI) unless the keyword NO-RI also appears in the command.

These clauses are generated for each DB2-TABLE member:

- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' sub-clause
- The COLUMNS CONTAINS clause of the DB2-TABLE member holds an entry for each column in the table. Column entries are separated by commas. Each entry includes the name of the column and, if the column is part of the primary key (that is, a prime column), the entry also includes the keyword NOT-NULL. The keyword PRIMARY-KEY is also included for each prime column unless NO-RI is specified in the command. There may be a CHECK-CONSTRAINT on the column, with an optional name and at least one CONDITION clause of up to 255 bytes. The CONDITION may be split over many 255 character strings, each prefixed by the CONDITION keyword.
- If the table contains any foreign keys and NO-RI has not been specified, a CONSTRAINT clause is generated for each relationship in which the table participates as a dependent table
- Each CONSTRAINT clause includes a FOREIGN-KEY clause with one or more entries, separated by commas, one per column of the foreign key. Each entry contains the name of the foreign key column and, if the foreign key relationship is of domain type, a MEMBER subclause which identifies the corresponding prime column in the parent table.

- A REFERENCES clause giving the name of the parent table also appears in the generated CONSTRAINT clause
- If specified in the DB2 PREVIEW command, a CREATOR-OWNER clause is included specifying a dictionary DB2-USER member as the creator or owner of the DB2 table
- If specified in the DB2 PREVIEW command, an IN tbspace-name clause is included, specifying the name of a dictionary DB2-TB SPACE member.

Refer to "Generated DB2-TABLE Definition" on page 85 for the syntax of the generated DB2-TABLE member.

Suppressing Support for Referential Integrity

To specify that the DB2-TABLE members must not contain clauses supporting referential integrity (RI), enter:

```
DB2 POPULATE TABLES NO-RI selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

This suppresses the PRIMARY-KEY keywords and foreign key CONSTRAINT clauses needed to support RI.

Generating References to Tablespaces

To generate a reference to a DB2-TBSPACE dictionary member in each generated DB2-TABLE dictionary definition, enter:

```
DB2 POPULATE TABLES TBSPACES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The name of the DB2-TBSPACE member is constructed from the name of the table concatenated with the suffix '-TBSP', unless you specify an alternative name or an alternative suffix (or prefix) in the command.

To specify a particular name which will appear in every DB2-TABLE member defined, enter:

```
DB2 POPULATE TABLES TESPACES NAME name selection ;
```

where *name* is a valid dictionary member name and selection is defined as above.

To specify a prefix or suffix to be concatenated with the table name, enter:

```
DB2 POPULATE TABLES TBSPACES PREFIX 'string' selection ;
```

or

```
DB2 POPULATE TABLES TBSPACES SUFFIX 'string' selection ;
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

Generating and Populating DB2-INDEX Members

To populate the dictionary with a DB2-INDEX member representing a primary index for each selected table, enter:

```
DB2 POPULATE INDEXES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The name of the DB2-INDEX member is constructed from the name of the table concatenated with the suffix '-IND', unless you specify an alternative suffix (or prefix) in the command.

To construct the DB2-INDEX name from the name of the table concatenated with a specified prefix or suffix, enter:

```
DB2 POPULATE INDEXES PREFIX 'string' selection ;
```

or

```
DB2 POPULATE INDEXES SUFFIX 'string' selection ;
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

These clauses are generated for each DB2-INDEX member:

- The CONTAINS clause of the DB2-INDEX definition holds an entry for each column in the primary key of the selected table
- The UNIQUE keyword is included, followed by an ON clause containing the name of the selected table. This indicates that the DB2-INDEX member represents a unique member.
- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' subclause
- If specified in the DB2 POPULATE command, a CREATOR-OWNER clause is also included which names a dictionary DB2-USER member as the creator or owner of the DB2 index.

Refer to "Generated DB2-INDEX Definition" on page 86 for the syntax of the generated DB2-INDEX definition.

Generating and Populating DB2-VIEW Members

To populate the dictionary with a DB2-VIEW member for each selected table, enter:

```
DB2 POPULATE VIEWS selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The DB2-VIEW member name is constructed from the name of the table concatenated with the suffix '-VIEW', unless you specify an alternative suffix (or prefix) in the command.

To construct the DB2-VIEW name from the name of the table concatenated with a specified prefix or suffix, enter:

```
DB2 POPULATE VIEWS PREFIX 'string' selection ;
```

or

```
DB2 POPULATE VIEWS SUFFIX 'string' selection ;
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

These clauses are generated for each DB2-VIEW member definition:

- The CONTAINS clause of the DB2-VIEW definition holds an entry for each column of the selected table
- The FROM clause contains a reference to the selected table
- The keywords SELECT ALL are included in the definition. They appear between the CONTAINS clause and the FROM clause and ensure that the SELECT ALL option will be included in the subselect clause of the SQL CREATE VIEW statement produced subsequently from the DB2-VIEW member.
- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' subclause
- If specified in the DB2 PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary DB2-USER member as the creator or owner of the DB2 view.

Refer to "Generated DB2-VIEW Definition" on page 87 for the syntax of the generated DB2-VIEW definition.

Generating References to a DB2 User

To specify that a selection of members generated from the WBDA belongs to a particular DB2-USER member, enter: `DB2 POPULATE` *`member-type-selection`* `CREATOR-OWNER` *`DB2-user selection`* `;`

where:

`member-type-selection` is one or more of the TABLES, INDEXES, and VIEWS clauses.

`DB2-user` is the name of a dictionary DB2-USER member.

`selection` is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

This causes a CREATOR-OWNER clause to be added to the generated dictionary members. If the DB2-USER member does not exist already in the dictionary, then a dummy member with that name is set up.

Generating and Populating a SYSTEM Member

To populate the dictionary with a SYSTEM member containing the names of all the DB2 members generated by this command, enter:

`DB2 POPULATE` *`member-type-selection`* `AS-SYSTEM` *`system-name selection`*`;`

`member-type-selection` is one or more of the TABLES, INDEXES and VIEWS clauses.

system-name is the name of the generated SYSTEM dictionary member.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The generated SYSTEM definition then is automatically added to the dictionary. The CONTAINS clause holds the names of all the dictionary members generated by the DB2 POPULATE command.

Refer to "Generated SYSTEM Definition" on page 88 for the syntax of the generated SYSTEM member.

Selecting Tables in the Workbench Design Area

When issuing the DB2 POPULATE command, you must specify which tables in the Workbench Design Area (WBDA) you want to use to generate dictionary definitions.

To specify all the tables in the WBDA enter:

```
DB2 POPULATE member-type-selection ALL ;
```

where *member-type-selection* is one or more of the TABLES, INDEXES, and VIEWS clauses.

The tables are selected in ascending order of their WBDA numbers unless ALPHABETICALLY also appears in the command.

To specify the tables by name, enter: `DB2 POPULATE member-type-selection NAMES name-list ;`

where:

member-type-selection is defined as above.

name-list is a list of one or more valid names of tables present in the WBDA.

Consecutive names must be separated by commas. The tables are selected in the order in which their names appear in name-list unless ALPHABETICALLY is also specified.

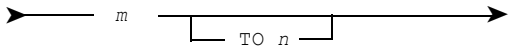
To specify the tables by WBDA number, enter:

```
DB2 POPULATE member-type-selection NUMBERS range-list ;
```

where:

member-type-selection is defined as above.

range-list is a list of one or more numeric ranges, separated by commas, each of the form:



where *m* and *n* are valid WBDA numbers and *n*, if it appears, is greater than *m*. Every table is selected whose WBDA number appears in the list or falls within a range appearing in the list. Definitions are generated and reported in the order listed unless the keyword ALPHABETICALLY is also specified in the command.

To specify that the tables are to be selected in alphanumeric order of table name, enter:

```
DB2 POPULATE member-type-selection selection ALPHABETICALLY ;
```

where:

member-type-selection is defined as above.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

Tailoring Generated Definitions

To generate and populate DB2 dictionary definitions in a format tailored to your requirements, enter: `DB2 POPULATE member-type-selection selection USING FORMAT format-member ;`

where:

member-type-selection is one or more of the TABLES, INDEXES. and VIEWS clauses.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

format-member is the name of a dictionary FORMAT member.

This outputs the dictionary definitions according to the specifications in the FORMAT member.

Use the USING FORMAT option to:

- Generate DB2 dictionary definitions compatible with any User Defined Syntax structure you may have implemented
- Generate dictionary member names conforming to your naming standards
- Generate dictionary definitions preceded by the REPLACE or INSERT command, instead of the default ADD command.

Combining DB2 POPULATE Command Options

You can generate and populate the dictionary with DB2 member definitions for any combination of the DB2-TABLE, DB2-INDEX, and DB2-VIEW member types in one DB2 POPULATE command, and optionally specify any or all of the CREATOR-OWNER, AS-SYSTEM, and USING-FORMAT clauses at the same time.

In addition to populating the dictionary with the generated member definitions, the DB2 POPULATE command provides a printout of the definitions.

If you want to generate definitions for more than one member type, you must specify the member type clauses in the command in the following order:

- TABLES clause
- INDEXES clause
- VIEWS clause

You must also include one of the ALL, NAMES, or NUMBERS clauses in the command to select the tables in the Workbench Design Area from which to generate the DB2 member definitions.

Examples of the DB2 POPULATE Command

To generate and populate a DB2-TABLE member:

- For a table named DEPARTMENT
- With no clauses for referential integrity (RI)
- Belonging to a system called DB2-SYSTEM-TEST

enter:

```
DB2 POPULATE TABLES NO-RI AS-SYSTEM DB2-SYSTEM-TEST NAMES DEPARTMENT
;
```

To generate and populate DB2-TABLE and DB2-INDEX members:

- For all the tables in the WBDA
- Including clauses to support RI
- Constructing each DB2-INDEX name from the table name concatenated with the suffix 'TEST'
- Specifying that the selected tables are to be processed in alphanumeric order of table name

enter:

```
DB2 POPULATE TABLES INDEXES SUFFIX 'TEST' ALL ALPHABETICALLY ;
```

To generate and populate DB2-INDEX and DB2-VIEW members:

- For tables in the WBDA selected by WBDA number
- With each DB2-INDEX definition name constructed from the table name concatenated with the default suffix '-IND'
- Constructing each DB2-VIEW definition name from the table name concatenated with the prefix 'TEST'
- Specifying that all the generated DB2-INDEX and DB2-VIEW definitions must reference a DB2-USER member named USER1

enter:

```
DB2 POPULATE INDEXES VIEWS PREFIX 'TEST' CREATOR-OWNER USER1
                                                NUMBERS 1 TO 3, 5 ;
```

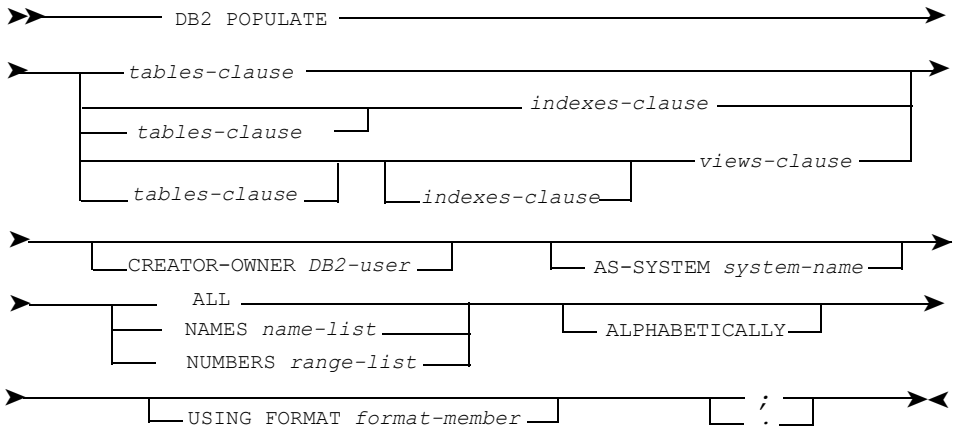
To generate and populate DB2-TABLE and DB2-VIEW members:

- For all tables in the WBDA,
- Suppressing clauses to support RI,
- Specifying that all the generated DB2-TABLE members must reference a DB2-TBSPACE member named DEP-TBSP, and
- Formatting the output according to a format definition named FMT-REPL,

enter:

```
DB2 POPULATE TABLES NO-RI TBSPACES NAME DEP-TBSP VIEWS ALL USING
                                                  FORMAT FMT-REPL ;
```

DB2 POPULATE Syntax



where *tables-clause* is:

```
►──── TABLES ┬──────────┬┬──────────────────────────────────────►
             └─NO-RI────┘└─ TBSPACES ──┬──────────────────────┬─┤
                                       ├── NAME name ─────────┤
                                       ├── PREFIX 'string' ───┤
                                       └── SUFFIX 'string' ───┘
```

where:

name is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

string is an alphanumeric string of up to 31 characters, conforming to the rules for a valid Manager Products repository member name.

indexes-clause is:

```
►►──── INDEXES ──┬──────────────────────┬──────────────►◄
                 ├── PREFIX 'string' ───┤
                 └── SUFFIX 'string' ───┘
```

where *string* is defined as above.

views-clause is:

```
►►──── VIEWS ──┬──────────────────────┬──────────────►◄
               ├── PREFIX 'string' ───┤
               └── SUFFIX 'string' ───┘
```

where *string* is defined as above.

where:

DB2-user is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

system-name is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

name-list is a list of validly named tables in the WBDA. If there are two or more names in the list they must be separated by commas.

range-list is a list of one or more numeric ranges, separated by commas, each of the form:

```
►──── m ──┬──────────┬──────────►
          └─ TO n ───┘
```

where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*.

format-member is the name of a previously defined, valid FORMAT member.

DB2 PREVIEW

DB2 PREVIEW generates and reports repository member definitions from DB2 tables, indexes, and views in the DB2 design.

Refer to "DB2 PREVIEW Syntax" on page 272 for the syntax of the DB2 PREVIEW command.

The DB2 PREVIEW command generates dictionary member definitions for one or more of the following member types, from selected tables of the DB2 design in the Workbench Design Area (WBDA):

- DB2-TABLE
- DB2-INDEX
- DB2-VIEW.

Each time you issue the command, a SYSTEM member also can be generated, containing a list of all the DB2 dictionary members generated by the command.

The command can be used only if the WBDA contains normalized data. If there is no data in the WBDA, or if the data has not been normalized, the command is terminated and a message to that effect is output.

The DB2 PREVIEW command will automatically generate the DB2 design if one has not already been generated. Any unnamed tables will be ignored by the command.

The generated definitions are not added to the dictionary, but are displayed so that you can check that they meet your database requirements. Each definition is preceded by an ADD command and followed by a terminator.

Once you are satisfied with the generated definitions, they can be added to the dictionary using the DB2 POPULATE command.

Although the content of a generated DB2 dictionary member definition is only a subset of the permissible content (which can be added later by a user if required), the generated definitions are complete enough to be used subsequently to produce valid SQL CREATE TABLE, CREATE INDEX, and CREATE VIEW statements. (When creating the DB2 object, DB2 will assign default values to all the remaining clauses.)

Each member definition is preceded by an ADD command and followed by a terminator.

DB2 PREVIEW automatically generates primary key keywords and foreign key clauses in DB2-TABLE definitions to support referential integrity, unless you use the NO-RI option to suppress them. In addition, the command enables you to assign DB2 tables to specific tablespaces (the TBSPACE option).

The command also allows you to associate a dictionary DB2-USER member (via the CREATOR-OWNER clause) with the DB2 dictionary members being defined.

You must enter one (and only one) of the following keywords or clauses in the command to indicate your selection of the DB2 tables in the WBDA to be used in generating the dictionary definitions:

- The ALL keyword to select all the tables in the WBDA
- The NAMES clause for a selection of tables by name
- The NUMBERS clause for a selection of tables by WBDA number.

If you also enter the keyword ALPHABETICALLY, the definitions are generated (and displayed) in alphanumeric order of table name.

The USING FORMAT clause allows you to specify a defined dictionary FORMAT member of the dictionary to control the format in which the definitions are generated. It is available only if you have the User Formatted Output facility installed.

You can generate dictionary definitions for any or all of the member types in one command, but you must specify them in the order in which the corresponding clauses appear in the command syntax.

Generating and Previewing DB2-TABLE Definitions

Use the keyword TABLES to generate and preview a DB2-TABLE dictionary definition for each selected table in the Workbench Design Area (WBDA). The name of the table in the WBDA becomes the name of the generated DB2-TABLE member.

To generate and preview DB2-TABLE definitions, enter:

```
DB2 PREVIEW TABLES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The command generates a DB2-TABLE definition for each selected table. The definition automatically contains PRIMARY-KEY keywords and foreign key CONSTRAINT clauses to support referential integrity (RI) unless you also specify the keyword NO-RI in the command to indicate that they are to be suppressed.

These clauses are generated for each DB2-TABLE member:

- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' subclause
- The COLUMNS CONTAINS clause of the DB2-TABLE member holds an entry for each column in the table. Column entries are separated by commas. Each entry includes the name of the column and, if the column is part of the primary key (that is, a *prime column*) the entry also includes the keyword NOT-NULL. The keyword PRIMARY-KEY is also included for each prime column unless NO-RI is specified in the command.
- CHECK-CONSTRAINT, if present on a column, may be named and will have a CONDITION clause - split into 255-character strings.
- If the table contains any foreign keys and NO-RI has not been specified, a CONSTRAINT clause is generated for each relationship in which the table participates as a dependent table
- Each CONSTRAINT clause includes a FOREIGN-KEY clause with one or more entries, separated by commas, one per column of the foreign key. Each entry contains the name of the foreign key column and, if the foreign key relationship is of domain type, a MEMBER subclause which identifies the corresponding prime column in the parent table.
- A REFERENCES clause giving the name of the parent table also appears in the generated CONSTRAINT clause
- If specified in the DB2 PREVIEW command, a CREATOR-OWNER clause is included specifying a dictionary DB2-USER member as the creator or owner of the DB2 table
- If specified in the DB2 PREVIEW command, an IN `tbspace-name clause` is included, specifying the name of a dictionary DB2-TBSPACE member.

Refer to "Generated DB2-TABLE Definition" on page 85 for the syntax of the generated DB2-TABLE definition.

Suppressing Support for Referential Integrity

To specify that the DB2-TABLE definitions must not contain clauses supporting referential integrity (RI), enter:

```
DB2 PREVIEW TABLES NO-RI selection ;
```

where `selection` is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

This suppresses the PRIMARY-KEY keywords and foreign key CONSTRAINT clauses needed to support RI.

Generating References to Tablespaces

To generate a reference to a DB2-TBSPACE dictionary member in each generated DB2-TABLE dictionary definition, enter:

```
DB2 PREVIEW TABLES TBSPACES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The name of the DB2-TBSPACE member is constructed from the name of the table concatenated with the suffix '-TBSP', unless you specify an alternative name or an alternative suffix (or prefix) in the command.

To specify a particular name which will appear in every DB2-TABLE member defined, enter:

```
DB2 PREVIEW TABLES TBSPACES NAME name selection ;
```

where:

name is a valid dictionary member name.

selection is defined as above.

To specify a prefix or suffix to be concatenated with the table name, enter:

```
DB2 PREVIEW TABLES TBSPACES PREFIX 'string'  selection ;
```

or

```
DB2 PREVIEW TABLES TBSPACES SUFFIX 'string' selection ;
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

Generating and Previewing DB2-INDEX Definitions

To specify that you want a DB2-INDEX definition, representing a primary index, to be generated for each selected table, enter:

```
DB2 PREVIEW INDEXES selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The name of the DB2-INDEX member is constructed from the name of the table concatenated with the suffix '-IND', unless you specify an alternative suffix (or prefix) in the command.

To construct the DB2-INDEX name from the name of the table concatenated with a specified prefix or suffix, enter:

```
DB2 PREVIEW INDEXES PREFIX 'string' selection
```

or

```
DB2 PREVIEW INDEXES SUFFIX 'string' selection
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

These clauses are generated for each DB2-INDEX definition:

- The CONTAINS clause of the DB2-INDEX definition holds an entry for each column in the primary key of the selected table
- The UNIQUE keyword is included, followed by an ON clause containing the name of the selected table. This indicates that the DB2-INDEX member represents a unique member.
- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' subclause
- If specified in the DB2 PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary DB2-USER member as the creator or owner of the DB2 index.

Generating and Previewing DB2-VIEW Definitions

To specify that you want a DB2-VIEW definition to be generated for each selected table, enter:

```
DB2 PREVIEW VIEWS selection ;
```

where *selection* is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The name of the DB2-VIEW definition is constructed from the name of the table concatenated with the suffix '-VIEW', unless you specify an alternative suffix (or prefix) in the command.

To construct the DB2-VIEW definition's name from the table name concatenated with a specified prefix or suffix, enter:

```
DB2 PREVIEW VIEWS PREFIX 'string' selection
```

or

```
DB2 PREVIEW VIEWS SUFFIX 'string' selection
```

where:

string is a valid dictionary string of up to 31 characters.

selection is defined as above.

These clauses are generated for each DB2-VIEW definition:

- The CONTAINS clause of the DB2-VIEW definition holds an entry for each column in the selected table
- The FROM clause contains a reference to the selected table
- The keywords SELECT ALL are included in the definition. They appear between the CONTAINS clause and the FROM clause and ensure that the SELECT ALL option will be included in the subselect clause of the SQL CREATE VIEW statement produced subsequently from the DB2-VIEW member.
- For each data-view which is the origin of a WBDA dependency represented by the table, the SEE clause contains a separate data-view FOR 'SOURCE' subclause
- If specified in the DB2 PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary DB2-USER member as the creator or owner of the DB2 view.

Refer to Chapter 9, "Repository Member Types," on page 341 for the syntax of the DB2-VIEW member type.

Generating References to a DB2 User

To specify that a selection of members generated from the Workbench Design Area (WBDA) belongs to a particular DB2-USER member, enter:

```
DB2 PREVIEW member-type-selection CREATOR-OWNER db2-user selection
;
```

where:

member-type-selection is one or more of the TABLES, INDEXES, and VIEWS clauses.

db2-user is the name of a dictionary DB2-USER member.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

This causes a CREATOR-OWNER clause to be added to the generated DB2 dictionary definitions. If the DB2-USER member does not exist already in the dictionary, then a dummy member is set up for that name.

Generating and Previewing a SYSTEM Definition

To generate and preview a SYSTEM definition containing the names of all the DB2 definitions generated by this command, enter:

```
DB2 PREVIEW member-type-selection AS-SYSTEM system-name selection ;
```

where:

member-type-selection is one or more of the TABLES, INDEXES, and VIEWS clauses.

system-name is the name of the generated SYSTEM dictionary definition.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

The CONTAINS clause in the generated SYSTEM definition holds the names of all the dictionary member definitions generated by this DB2 PREVIEW command.

Refer to Chapter 3, "Generated SYSTEM Definition," on page 88 for the syntax of the SYSTEM member type generated by the DB2 PREVIEW command.

Selecting Tables in the Workbench Design Area

When issuing the DB2 PREVIEW command, you must specify the tables in the Workbench Design Area (WBDA) from which you want to generate dictionary definitions for previewing.

To specify all the tables in the WBDA, enter:

```
DB2 PREVIEW member-type-selection ALL ;
```

where *member-type-selection* is one or more of the TABLES, INDEXES, and VIEWS clauses.

The tables are selected in ascending order of their WBDA numbers unless ALPHABETICALLY also appears in the command.

To specify the tables by name, enter:

```
DB2 PREVIEW member-type-selection NAMES name-list ;
```

where:

member-type-selection is defined as above.

name-list is a list of one or more valid names of tables present in the WBDA. Consecutive names must be separated by commas. The tables are selected in the order in which their names appear in name-list unless ALPHABETICALLY is also specified.

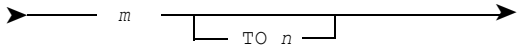
To specify the tables by WBDA number, enter:

```
DB2 PREVIEW member-type-selection NUMBERS range-list ;
```

where:

member-type-selection is defined as above.

range-list is a list of one or more numeric ranges, separated by commas, each of the form:



where *m* and *n* are valid WBDA numbers and *n*, if present, is greater than *m*. Every table is selected whose WBDA number appears in the list or falls within a range appearing in the list. Definitions are generated and reported in the order listed unless the keyword ALPHABETICALLY is also given in the command.

To specify that the tables are to be selected in alphanumeric order of table name, enter:

```
DB2 PREVIEW member-type-selection selection ALPHABETICALLY ;
```

where:

member-type-selection is defined as above.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

Tailoring Generated Dictionary Definitions

To generate and preview DB2 dictionary definitions in a format tailored to your requirements, enter:

```
DB2 PREVIEW member-type-selection selection USING FORMAT
format-member ;
```

where:

member-type-selection is one or more of the TABLES, INDEXES, and VIEWS clauses.

selection is one of the ALL, NAMES, or NUMBERS options used to identify the tables from which dictionary definitions are to be generated.

format-member is the name of a dictionary FORMAT member.

This outputs the dictionary definitions according to the specifications in the FORMAT member.

Use the USING FORMAT option to:

- Generate DB2 dictionary definitions compatible with any User Defined Syntax structure you may have implemented
- Generate member names to conform to your own naming standards
- Generate dictionary definitions preceded by the REPLACE or INSERT command, instead of the default ADD command.

Combining DB2 PREVIEW Command Options

You can generate dictionary definitions for any combination of the DB2-TABLE, DB2-INDEX, and DB2-VIEW member types in one DB2 PREVIEW command, and optionally specify any or all of the CREATOR-OWNER, AS-SYSTEM, and USING-FORMAT clauses at the same time. If you want to generate definitions for more than one member type, you must specify the member type clauses in the command in the following order:

- TABLES clause
- INDEXES clause
- VIEWS clause

You must also include one of the ALL, NAMES, or NUMBERS clauses in the command, to select the tables in the Workbench Design Area from which to generate the DB2 member definitions.

Examples of the DB2 PREVIEW Command

To generate and preview a DB2-TABLE definition:

- For a Workbench Design Area (WBDA) table named DEPARTMENT
- With no clauses for referential integrity (RI)
- Belonging to a system called DB2-SYSTEM-TEST

enter:

```
DB2 PREVIEW TABLES NO-RI AS-SYSTEM DB2-SYSTEM-TEST NAMES DEPARTMENT
;
```

To generate and preview DB2-TABLE and DB2-INDEX definitions:

- For all the tables in the WBDA,
- Including clauses to support RI,
- Constructing each DB2-INDEX name from the table name concatenated with the suffix 'TEST', and
- Specifying that the selected tables are to be processed in alphanumeric order of table name,

enter:

```
DB2 PREVIEW TABLES INDEXES SUFFIX 'TEST' ALL ALPHABETICALLY ;
```

To generate and preview DB2-INDEX and DB2-VIEW definitions:

- For tables in the WBDA selected by WBDA number, with each DB2-INDEX definition name constructed from the table name concatenated with the default suffix '-IND',
- Constructing each DB2-VIEW definition name from the table name concatenated with the prefix 'TEST', and
- Specifying that all the generated DB2-INDEX and DB2-VIEW definitions must reference a DB2-USER member named USER1,

enter:

```
DB2 PREVIEW INDEXES VIEWS PREFIX 'TEST' CREATOR-OWNER USER1
                                                NUMBERS 1 TO 3, 5 ;
```

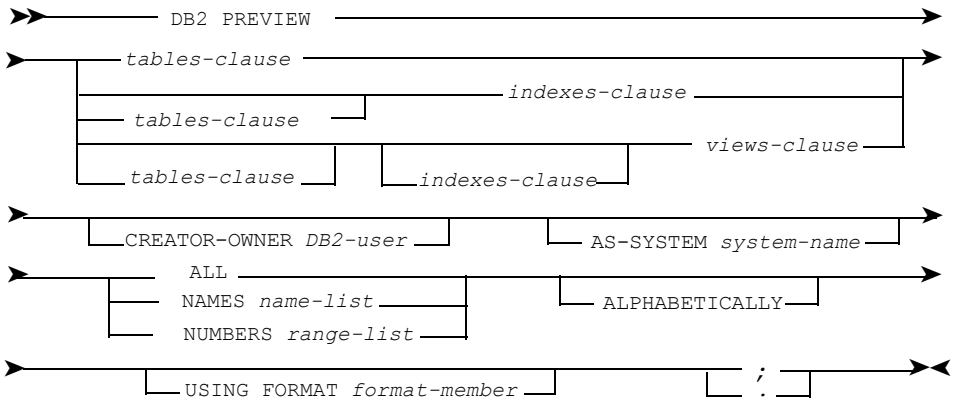
To generate and preview DB2-TABLE and DB2-VIEW definitions:

- For all tables in the WBDA,
- Suppressing clauses to support RI,
- Specifying that all the generated DB2-TABLE definitions must reference a DB2-TBSPACE definition named DEP-TBSP, and
- Formatting the output according to a format definition named FMT-REPL,

enter:

```
DB2 PREVIEW TABLES NO-RI TBSPACES NAME DEP-TBSP VIEWS ALL
                                          USING FORMAT FMT-REPL ;
```

DB2 PREVIEW Syntax



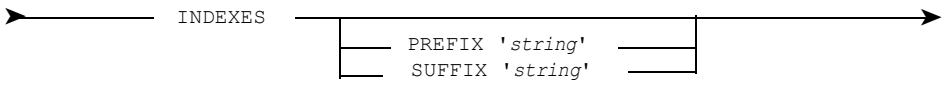
where *tables-clause* is:

where:

name is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

string is an alphanumeric string of up to 31 characters, conforming to the rules for a valid Manager Products repository member name.

indexes-clause is:



where *string* is defined as above.

views-clause is:

```
➤──── VIEWS ──┬──────────────────────────┬──────────────➤
              │  PREFIX 'string'         │
              │  SUFFIX 'string'         │
```

where *string* is defined as above.

where:

DB2-user is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

system-name is an alphanumeric string of up to 32 characters, conforming to the rules for a valid Manager Products repository member name.

name-list is a list of validly named tables in the WBDA. If there are two or more names in the list they must be separated by commas.

range-list is a list of one or more numeric ranges, separated by commas, each of the form:

```
➤──── m ──┬──────────┬──────────➤
          │ TO n     │
```

where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*.

format-member is the name of a previously defined, valid FORMAT member.

DB2 PRODUCE

DB2 PRODUCE generates either a host language data structure or a table layout from a DB2-TABLE or DB2-VIEW repository member.

Refer to "DB2 PRODUCE Syntax" on page 280 for the syntax of the DB2 PRODUCE command.

Use the DB2 PRODUCE command to generate either:

- A host language data structure in Assembler language, PL/1, or COBOL, or
- A table layout describing a table or view documented by repository members

by entering either:

```
DB2 PRODUCE language FROM member ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member ;
```

where:

`language` is ASSEMBLER (or ALC, ASM or BAL), PL1 (or PLI, PL/I or PL/l) or COBOL.

`member` is the name of a DB2-TABLE or DB2-VIEW repository member.

Column variables in the host language data structure are generated from the members named in the CONTAINS attribute of the relevant DB2-TABLE or DB2-VIEW member. Host language data types are generated for these variables, corresponding to the DB2 data type of columns in the table or view.

The generated host (and indicator) structures can be referenced by application programs containing embedded SQL syntax for data manipulation statements.

You can generate (flat), two-level data structures for use within SQL statements (as these conform with the DB2 language pre-processors), or (nested) multi-level data structures (useful in COBOL or PL1 programs).

You can automatically generate SQL DECLARE TABLE statements with your host language data structure or table layout

Generated host language data structures are displayed on the screen. You can tailor this output, file it on the MP-AID, or send it to an external file, using output generation options. You can also tailor output by calling executive routines (user exit routines) at set points (user exits) during output. This process is known as taking user exits.

The systems administrator can tailor output by altering the DB2 profile.

Refer to Chapter 4, "Repository Definition," on page 91 for details of generating column data types.

Refer to Chapter 5, "Export to DB2," on page 105 for details of user exits, and the DB2 profile.

A table layout displays information about a table or view by listing the following repository members documenting the structure of that object:

- The DB2-TABLE or DB2-VIEW members (documenting the table/view)
- The GROUP and ITEM members (documenting the columns in the table/view)
- The PROGRAM, MODULE, SYSTEM, and MMR-SYSTEM members (documenting the constraints, edit procedures, field procedures and validation procedures of a table).

All attributes of the relevant DB2-TABLE or DB2-VIEW member are given. Information such as the DB2 data type and length of columns is also displayed.

By printing the table layouts you can produce paper documentation displaying both the structure and purpose of your tables and views.

Description of Table Layouts For DB2 Tables

These column headings are given in table layouts for DB2 tables:

PKEY. The column is documented in the repository as being a primary key if Y is specified and is not a primary key if there is no entry. If the position (and sequence) is documented in the definition, then the position is shown, followed by either (A) for an ascending sequence, or (D) for a descending sequence.

NULL. The column is defined as not null if N is specified, not null with default values if D is specified, and nullable if there is no entry.

NAME. The names of the members documenting the table and their columns and edit, field, and validation procedures.

TYPE. For the member documenting the table or view, TYPE is either DB2-TABLE or DB2-VIEW. Otherwise, TYPE is either:

- The DB2 data type of a column, if the member is an ITEM or GROUP member documenting the column
- EDITPROC, FIELDPROC or VALIDPROC, if the member documents an edit, field or validation procedure

LENGTH. The maximum number of bytes to store a value of this column; either recorded directly in an ITEM member, or the sum of all contained ITEM lengths in a GROUP member. The length of the row for the table equals the total length of all the columns. The length is increased by one byte for nullable columns.

REMARKS. The text extracted using the GIVING keyword, plus any referential constraints for that table.

At the end of the table layout, the contents of the IN attribute in the DB2-TABLE member are given, in the REMARKS columns.

Description of Table Layouts For DB2 Views

These column headings are given in table layouts for DB2 views:

GRP-BY. A Y shows that the GROUP-BY attribute in the DB2-VIEW member has been specified; otherwise, the column is blank. Also given in this column are the contents of any WHERE and HAVING attributes in the DB2-VIEW member, at the end of the table layout.

The next column heading in the table layout is blank. In this column, CORR may be displayed for a column, if relevant. If this keyword appears, then the correlation name itself is given in the next column.

WHERE, HAVING, and FROM. May also be displayed in the second column, at the end of the table layout, if relevant Details are then shown in the third column (NAME).

NAME, TYPE, LENGTH, REMARKS. As for DB2 tables.

Example of a Table Layout

Figure 34 is an example of a table layout generated from a DB2-TABLE member:

Figure 34 • Table Layout Generated from a DB2-TABLE Member

```
*******************************************************************************************
Description Of TB2-Employee
*******************************************************************************************
PKEY      NULL    NAME             TYPE             LENGTH       REMARKS
*******************************************************************************************
                  TB-DJB-EMPS      DB2-TABLE        57
          N       IT-EMPNO         CHAR(6)          6
Y         N       IT-MIDINIT       CHAR(2)          2
          N       IT-LASTNAME      VARCHAR(10)      12           Surname.
                  IT-JOB           CHAR(8)          9            Duties.
                  IT-SALARY        DECIMAL(9,2)     6
                  MO-FPO2          FIELDPROC
                  MO-VLO3          VALIDPROC
                  MO-EP21          EDITPROC
```

Expanding Nested Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for this member, you should use the same keywords for this command, by entering either:

```
DB2 PRODUCE language FROM member EXPAND ;
```

or

```
DB2 PRODUCE language FROM member NO-EXPAND ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member EXPAND ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member NO-EXPAND ;
```

EXPAND clauses in the DB2-TABLE member are used as the default.

Refer to "DB2 CREATE" on page 212 for use of the EXPAND/NO-EXPAND keywords.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for further details about the EXPAND clause.

Taking User Exits

To take a user exit, enter either:

```
DB2 PRODUCE language FROM member USING exit-routine ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member USING exit-routine ;
```

where *exit-routine* is the name of an executive routine.

Note:

The systems administrator can alter your DB2 profile so that a default user exit is always taken with the DB2 PRODUCE command. The USING keyword will override any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Generating SQL DECLARE TABLE Statements

To generate SQL DECLARE TABLE statements in addition to a host language data structure, enter:

```
DB2 PRODUCE language FROM member-name WITH-DECLARE ;
```

This has the effect of executing separate DB2 PRODUCE and DB2 DECLARE commands. However, using the WITH-DECLARE keyword is more efficient.

To specify an owner for the SQL DECLARE TABLE statement (overriding any owner defined in the member), enter:

```
DB2 PRODUCE language FROM member SQLID owner WITH-DECLARE ;
```

where *owner* is a delimited string of up to 8 characters, giving the ID of a specific user. To specify a location for the SQL DECLARE TABLE statement (overriding any location defined for that member), enter either:

```
DB2 PRODUCE language FROM member LOCATION loc-id WITH-DECLARE ;
```

where *loc-id* is a delimited string of up to 8 characters, giving the object's location.

Note:

You can only generate a DECLARE statement with a corresponding host language data structure. Therefore, you cannot use the WITH-DECLARE keyword with the TABLE-LAYOUT option of the DB2 PRODUCE command.

Generating Flat or Nested Data Structures

To generate or display flat (two-level) data structures, enter either:

```
DB2 PRODUCE language FROM member FOR SQL ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member FOR SQL ;
```

These structures are valid for use as host structures within SQL statements. For COBOL, an extra level is generated for variable length fields as required by DB2.

To show intermediate levels as comments, use the SHOW-INTERMEDIATE-LEVELS keyword after the FOR SQL keyword.

To generate indicator structures for corresponding generated host language data structures, use the INDICATOR keyword. To generate these structures as arrays, also use the ARRAY keyword; alternatively, to generate them as structures which match the main host structure, use the STRUCTURE keyword.

For example, to generate a flat COBOL data structure from a DB2-TABLE member named TB-DJB-EMPS, generating a corresponding indicator structure as an array, enter:

```
DB2 PRODUCE COBOL FROM TB-DJB-EMPS FOR SQL INDICATOR ARRAY ;
```

Note:

Only use the INDICATOR keyword when generating host language data structures--it has no purpose when generating a table layout.

To generate or display nested (multi-level) data structures, enter either:

```
DB2 PRODUCE language FROM member FOR work-store ;
```

or

```
DB2 PRODUCE TABLE-LAYOUT FROM member FOR work-store ;
```

where *work-store* is either WORKING-STORAGE or WS.

You may alter the suffixes generated by the PRODUCE COBOL command by changing the contents of these variables:

- MPDY_IND_SUFFIX contains the suffix on generated indicator structure names
- MPDY_JND_ARRAY_SUFF_1 and MPDY_IND_ARRAY_SUFF_2 contain the suffixes applied to the names generated for indicator arrays
- MPDY_COB_SUFF_1 and MPDY_COB_SUFF 2 contain the suffixes applied to the names generated in the hist structure for a column that has a variable length.

Refer to Chapter 5, "Export to DB2," on page 105 for further details of these profile variables.

Adding Additional Information to the Output

Use the GIVING keyword in conjunction with the USING keyword and a named exit to put the contents of NOTE, DESCRIPTION, or COMMENT in comments in the relevant language, when generating host language data structures by entering:

```
DB2 PRODUCE language FROM member GIVING attribute USING exec;
```

where *attribute* is the name of any text attribute.

The text attributes for which the exit provided by ASG caters are NOTE, DESCRIPTION, and COMMENT. The user is free to amend this exit. The exit will also need to be amended should the user want to put these text attributes or others into the remarks column when generating a table layout.

This combination of keywords allows you to describe DB2 objects in greater detail than would otherwise be possible.

Name Editing Options

Use the REPLACE/REPLACING, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

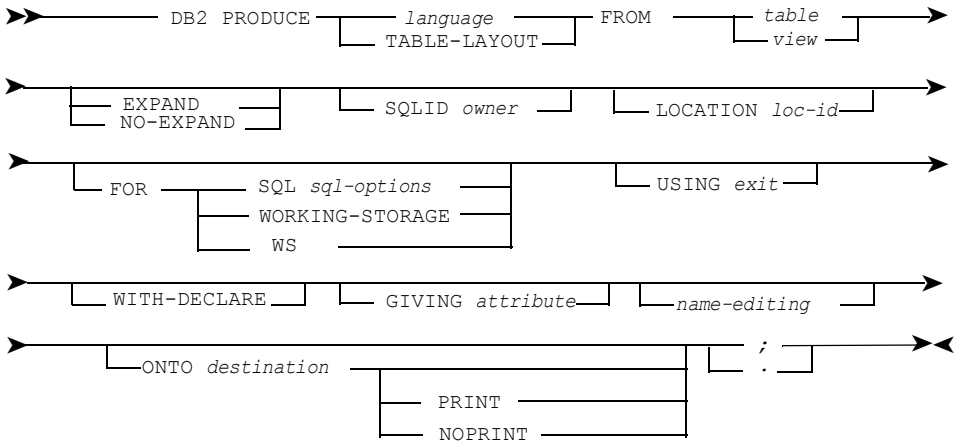
Output Generation Options

Use the ONTO keyword to direct your generated output to a specific destination. This destination can be:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

Refer to "Output Generation Options" on page 334 for further details of output generation options.

DB2 PRODUCE Syntax



where:

language is:

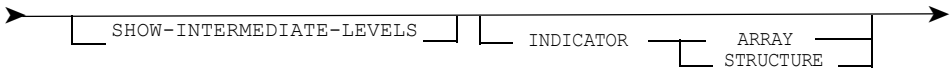
table is the name of a DB2-TABLE repository member.

view is the name of a DB2-VIEW repository member.

owner is a delimited string of up to 8 characters, giving the authorization ID of a specific user (for an SQL DECLARE statement).

loc-id is a delimited string of up to 16 characters, giving the name of a DB2 location (for an SQL DECLARE statement).

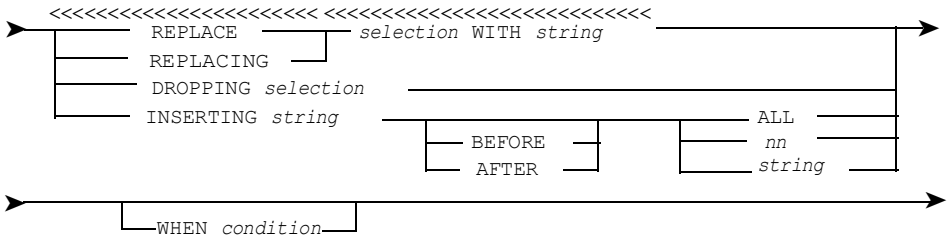
sql-options are:



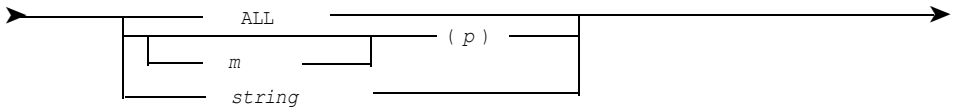
attribute is any text attribute.

user-exit is the name of an executive routine.

name-editing is:



where *selection* is:



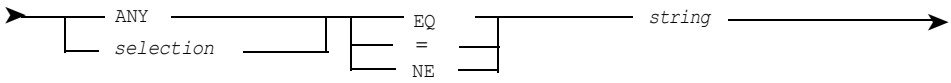
where:

m and *p* are integers in the range 1 to 96.

string is a delimited string of not more than 32 printable characters.

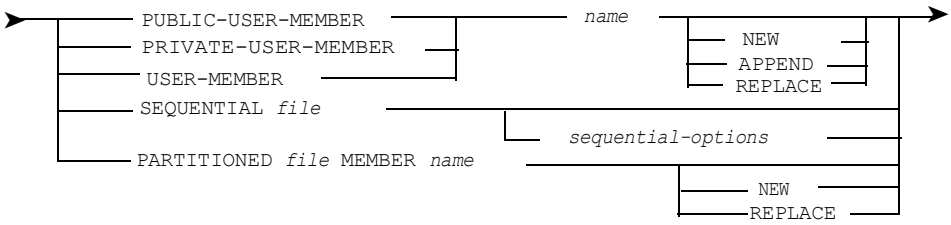
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

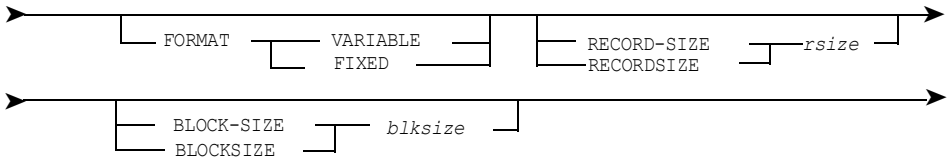


where:

name is the name of a USER-MEMBER.

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length.

blksize is the block size.

DB2 RECALCULATE

DB2 RECALCULATE recalculates sizes of tables or indexes.

Refer to "DB2 RECALCULATE Syntax" on page 284 for the syntax of the DB2 RECALCULATE command.

Use the DB2 RECALCULATE command immediately after using the DB2 SIZE command to recalculate the size of a DB2 table or index, based on altered specified values (such as cardinality). The report obtained is identical in form to the report from the DB2 SIZE command.

Using DB2 RECALCULATE is faster than re-issuing a DB2 SIZE command. You can therefore experiment more to better understand the impact of different values on object size.

You can only recalculate sizes immediately after entering a DB2 SIZE command. You can recalculate sizes any number of times.

To recalculate a table or index size, enter:

```
DB2 RECALCULATE changes ;
```

where *changes* specify the values you wish to override. If you specify no changes, the same report as from the previous DB2 SIZE command is given.

Refer to the following items for details of the changes you can specify.

Recalculated values are reported with (Command) beside them. Where possible, values not given in the command or the member definition default to IBM standard values.

Just as with the DB2 SIZE command, you can obtain a detailed or summary report, and add to the report using user exits, by using the SUMMARY, DETAIL, and USING keywords respectively.

Specifying Row Sizes, Cardinality, and Free Pages

To recalculate table or index sizes based on changed minimum or maximum row size, cardinality, number of free pages, or percentage of free pages, use the MINIMUM-ROW-SIZE, MAXIMUM-ROW-SIZE, CARDINALITY, FREEPAGE, or PCTFREE keywords followed by a new value for each.

For example, to recalculate a size of the DB2 table documented by the member TB-DJB-CUST, with a minimum row size of 6, enter:

```
DB2 RECALCULATE MINIMUM-ROW-SIZE 6 ;
```

Specifying an Edit Routine or a Bufferpool

For table size recalculations, you can choose whether an edit routine is to be used (involving an overhead of 10 bytes), and you can also select a specific bufferpool.

To specify whether or not an edit routine is to be used, enter either:

```
DB2 RECALCULATE EDITPRDC YES ;
```

or

```
DB2 RECALCULATE EDITPROC NO ;
```

By default (if you do not use this keyword), no edit routines are used.

To specify a bufferpool, enter:

```
DB2 RECALCULATE BUFFERPOOL buff-pool ;
```

where *buff-pool* is the bufferpool name: BP0, BP1, BP2, (for 4K page sizes) or BP32K (for 32K page sizes).

Specifying Subpages and Duplicate Key Values

For index size recalculations, you can specify a number of subpages, and an estimated number of duplicate key values. Duplicate key values are not given in the member definition, and default to a value of one. Only integral values may be specified.

To recalculate an index size based on an altered number of subpages, enter:

```
DB2 RECALCULATE SUBPAGES subpgs ;
```

where:

index is the name of a DB2-INDEX repository member.

subpgs is the number of subpages; either 1, 2, 4, 8, or 16.

To recalculate an index size based on an estimated number of duplicate key values, enter:

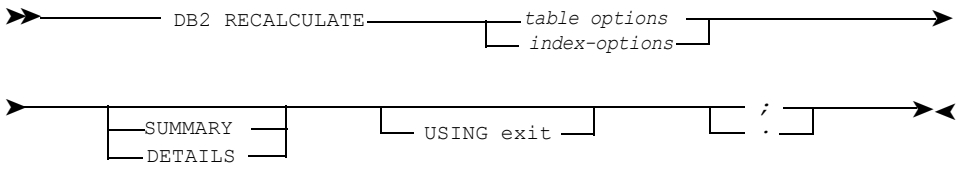
```
DB2 RECALCULATE DUPLICATES duplics ;
```

where *duplics* is the estimated number of duplicate key values.

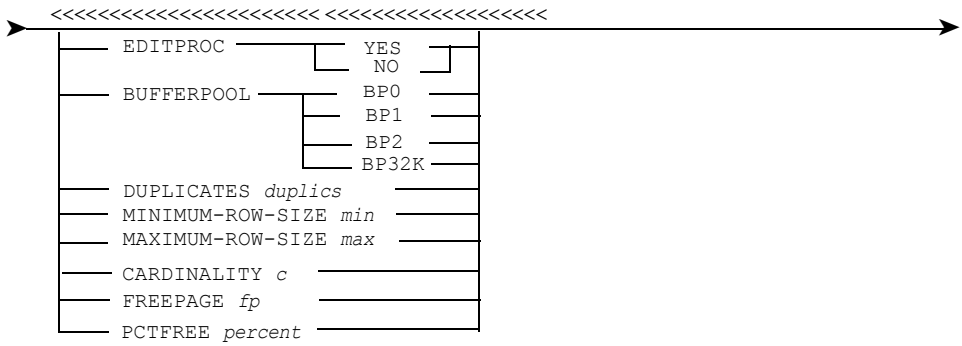
DB2-INDEX Member Types

For DB2-INDEX member types, the DB2 RECALCULATE command will give fixed length column results which may also be specified with the MINIMUM-ROW-SIZE keyword.

DB2 RECALCULATE Syntax



where *table-options* are:



where *min*, *max*, *c*, *fp*, and *percent* are integers, with *percent* being between 0 and 100.

index-options are:

```
<<
```

For example, a table called TESTTAB with an SQL alias of MYTAB is defined on the repository as:

```
DB2-TABLE
  CREATOR-OWNER MYSELF
  ALIAS
    SQL "MYTAB"
  IN MYDB
  COLUMNS HELD AS
     CONTAINS C1, C2, C3
```

Example 1 - the following command to rename an alias name is entered on May 28, 2002:

```
DB2 RENAME TESTTAB ALIAS SQL TO HISTAB
```

The output generated from execution of this command is:

```
AMEND TESTTAB
  APPEND RENAMED 'MYTAB' DATE 28.MAY.2002
  REPLACE OCCURRENCE 1 ALIAS SQL      "HISTAB"
;
RENAME TABLE MYSELF.MYTAB
          TO MYSELF.HISTAB
;
```

When the AMEND command is executed, the repository member TESTTAB is altered as follows:

```
DB2-TABLE
  CREATOR-OWNER MYSELF
  ALIAS
    SQL "HISTAB"
  IN MYDB
  COLUMNS HELD AS
     CONTAINS C1, C2, C3
  RENAMED MYTAB DATE 28/May/2002
```

Example 2 - the following command to rename the member is entered on May 28, 2002:

```
DB2 RENAME TESTTAB TO HIS-TAB
```

The output generated from execution of this command is:

```
AMEND TO
  APPEND RENAMED 'MYTAB' DATE 28.MAY.2002
;
RELABEL TESTTAB HIS-TAB ;
RENAME TABLE MYSELF.MYTAB
          TO MYSELF.HIS_TAB
;
```

When the AMEND command is executed, the RENAMED line is appended to the DB2_TABLE definition in the repository.

When the RELABEL command is executed, the DB2-TABLE TESTTAB and all references to it are renamed to HIS-TAB (see the RELABEL command in the *ASG-MethodManager User's Guide*).

DB2-RENAME Syntax

```
>>-- DB2-RENAME old-table-name --+-----------------------+-- TO new-table-name -->
                                 +-- ALIAS alias-type ---+

>--+-- ALIAS alias-type --+--------------------------------------------; --><
   +-- USING exit-routine -+                                           .
```

where:

old-table-name is the member name of the DB2-TABLE being renamed.

alias-type is the alias type of the alias name that is being renamed. The alias types are defined by your system administrator during installation.

new-table-name is the new name to be given the DB2-TABLE. If an alias name is being changed, then this is the new alias name to be used.

new-alias-type is the new alias type to be assigned to the new alias name.

exit-routine is the name of the executive routine that is to be executed when the member name is being renamed to make whatever modifications may be necessary to meet the naming standards of your installation. See Chapter 5, "Export to DB2," on page 105 for information on writing user exit executive routines.

Note:

If RENAME will modify member names in the repository, you must use the MethodManager command RELABEL; otherwise this option is not available for Manager Products's BLT.

DB2 REPORT

DB2 REPORT produces a DB2 Table Report of all or some of the tables in the DB2 design.

Refer to "DB2 REPORT Syntax" on page 291 for the syntax of the DB2 REPORT command.

Use the DB2 REPORT command to produce a DB2 Table Report of all or some of the tables in the DB2 design generated in the Workbench Design Area (WBDA).

You must enter one (and only one) of the following keywords or clauses in the command to indicate your selection of the tables to be reported:

- The ALL keyword to select all the tables in the WBDA
- The NAME clause for a selection of tables by name
- The NUMBERS clause for a selection of tables by number.

If you also enter the keyword ALPHABETICALLY, the selected tables will be output alphanumerically.

For each selected table in the WBDA, the report describes the dependencies represented by the table and the other tables to which it is related.

The command can be used only if the WBDA contains normalized data. If there is no data in the WBDA, or if it has not been normalized, you are informed and the command is terminated. If the WBDA contains normalized data but no DB2 design, the command causes the DB2 design to be generated and then produces the report.

The USING FORMAT option of this command is available only if you have the User Formatted Output facility installed. It allows you to specify the name of a valid FORMAT member of the dictionary in order to tailor the format in which the tables are output.

Refer to "Output from the DB2 REPORT Command" on page 54 for details of the DB2 REPORT command output.

Reporting All the Tables in the Workbench Design Area

To report all the tables in the Workbench Design Area (WBDA), enter:

```
DB2 REPORT TABLES ALL ;
```

This outputs the tables in order of WBDA number. To report all the tables alphanumerically, enter:

```
DB2 REPORT TABLES ALL ALPHABETICALLY ;
```

This causes the named tables to be reported in alphanumeric order of table name, followed by any unnamed tables in ascending order of WBDA number.

Reporting Tables Selected by Name

To report tables selected by name, enter:

```
DB2 REPORT TABLES NANES name-list ;
```

where *name-list* is a list of one or more valid names of tables present in the Workbench Design Area (WBDA). Table names in *name-list* must be separated by commas.

Tables are reported in the order listed unless the keyword ALPHABETICALLY also is specified in the command.

To report the tables in alphanumeric order of table name, enter:

```
DB2 REPORT TABLES NAMES name-list ALPHABETICALLY ;
```

For example:

```
DB2 REPORT TABLES NAMES DEPARTMENT, OFFICE, EMPLOYEE ALPHABETICALLY
;
```

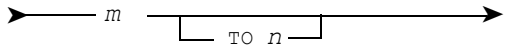
Reporting Tables Selected by Number

Tables in the Workbench Design Area (WBDA) which have not yet been named can be selected only by number.

To report tables selected by their WBDA number, enter:

```
DB2 REPORT TABLES NUMBERS range-list ;
```

where *range-list* is a list of one or more numeric ranges, separated by commas, each of the form:



where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*. Every table is reported whose WBDA number appears in the list or falls within a range appearing in the list. Tables are reported in the order listed unless the keyword ALPHABETICALLY is also specified in the command.

To report the listed tables alphanumerically, enter:

```
DB2 REPORT TABLES NUMBERS range-list ALPHABETICALLY ;
```

This causes the named tables in range-list to be reported in alphanumeric order of table name, followed by any unnamed tables in ascending order of WBDA number.

An example of this option is shown below:

```
DB2 REPORT TABLES NUMBERS 1,4,6 TO 12,17 TO 20,25 ALPHABETICALLY ;
```

Reporting Tables in a Specific Format

To report tables in a format tailored to your requirements, enter:

```
DB2 REPORT TABLES selection USING FORMAT format-member ;
```

where:

format-member is the name of a previously defined FORMAT member of the dictionary. Tables are output according to the specifications in the FORMAT member

selection is one of these:

- ALL
- NAMES *name-list*
- NUMBERS *range-list*

DB2 REPORT Syntax

```
>>──── DB2 REPORT TABLES ──┬─ ALL ──────────────────┬──────────────────>
                          ├─ NAMES name-list ──────┤
                          └─ NUMBERS range-list ───┘
>──┬──────────────────────┬──────────────────────────────────────────>
  └─ ALPHABETICALLY ─────┘
>──┬──────────────────────────────────────────┬──┬─ ; ─┬──────────>>
  └─ USING FORMAT format-member ─────────────┘  └─ . ─┘
```

where:

name-list is a list of validly named tables in the WBDA. If there are two or more names in the list they must be separated by commas

range-list is a list of one or more numeric ranges, separated by commas, each of the form:

```
>──── m ──┬──────────┬──────────>
         └─ TO n ───┘
```

where *m* and *n* are valid WBDA table numbers and *n*, if it appears, is greater than *m*.

format-member is the name of a previously defined, valid FORMAT member.

DB2 SIZE

DB2 SIZE calculates the size of a DB2 table or an index from its repository member definition.

Refer to "DB2 SIZE Syntax" on page 294 for the syntax of the DB2 SIZE command.

To give a report estimating the size of a table or an index in your DB2 environment, from a DB2-TABLE or a DB2-INDEX repository member, enter: `DB2 SIZE member ;`

where *member* is the name of a DB2-TABLE or DB2-INDEX repository member.

The report gives information about the size of the table or index specified. Extra terms are used in the "source" column:

Forced	If the page size given in the member was too small, it has been forced to 32K.
Assumed	No specified value was found, so the value given is the default assumed.
Member	The value shown is from an attribute in a repository member.
Command	The value shown is from a keyword in the command.

Use the report to determine the amount of primary or secondary storage space needed for a tablespace (defined in the PRIQTY and SECQTY attributes of DB2-TBSPACE members), and plan for future data growth.

Examine values and add extra details to the report by calling executive routines (*user exit routines*) at set points (*user exits*) during output. The basic report can only be added to not changed.

Refer to "Tailoring Output" on page 109 for details of user exits.

How Sizes are Calculated

Calculations take into account all factors affecting a table or index size, including page sizes (determined from bufferpool sizes), PCTFREE and FREEPAGE values, cardinality and other factors.

Table or index sizes are given in kilobytes. Row sizes are given in bytes.

Some calculations are given in a minimum...maximum range, whereas others are in a maximum...minimum range. This is because some calculations are inversely related to others, so a maximum for one corresponds to a minimum for another.

Table size is estimated by calculating the minimum and maximum size of each row in the table, and using the number of rows the table contains. The size of a row is given by the DB2 data type of the columns in the table (given in the relevant ITEM and GROUP members). Table size estimates give tablespace size estimates, as the size of a tablespace is equal to the size of the contained table or tables.

The size of a physical row can vary in practice if it contains any variable length columns. The estimated size uses minimum and maximum sizes for all variable columns, and so gives the range of sizes for the whole row.

If the column can contain null values (that is the keywords NOT-NULL and WITH-DEFAULT have not been specified in the DB2-TABLE member's definition) it is given an extra byte.

The number of pages required for the table is calculated from the row size according to the formulas given in the IBM DB2 documentation.

For size calculations, a column with a DB2 data type of DATE requires four bytes of storage, a column with a DB2 data type of TIME three bytes of storage, and a column with a DB2 data type of TIMESTAMP 10 bytes of storage.

Refer to "Documenting DB2 Security Information" on page 96 for details of generating column data types.

For index calculations, the output also shows how many pages are required for each level of the (b-tree) index, up to the root page.

Choosing a Summary or a Detailed Report

To obtain a summary report (the default), enter:

```
DB2 SIZE member SUMMARY ;
```

where *member* is a DB2-TABLE or DB2-INDEX repository member.

To obtain a detailed report, giving the name, data type, and minimum and maximum sizes for each column, enter:

```
DB2 SIZE member DETAILS ;
```

Taking User Exits

To take a user exit, enter:

```
DB2 SIZE member USING exit-routine ;
```

where:

member is the name of a DB2-TABLE or DB2-INDEX repository member.

exit-routine is the name of an executive routine.

Refer to "Tailoring Output" on page 109 for details of tailoring generated output.

Specifying Which Columns in an Indexed Table are Nullable

You can obtain greater accuracy in estimating index sizes by knowing which key columns in the indexed table are nullable (as nullable columns have an extra byte). A DB2-INDEX member does not show whether an indexing column is nullable or not. This is given in the definition of the table being indexed.

To accurately determine which columns are nullable, specify that you wish to look up the NOT NULL attribute in the indexed table, by entering:

```
DB2 SIZE member NOT-NULL REFER ;
```

where *member* is the name of a DB2-INDEX repository member.

This option requires more processing time. You can prevent this look-up process, and so increase speed, if you know that columns are all either nullable or not nullable. To specify that all indexed columns are nullable, enter:

```
DB2 SIZE member NOT-NULL NONE ;
```

or if all columns are not nullable (the default), enter:

```
DB2 SIZE member NOT-NULL ALL ;
```

Calculations Based On Expanded Data Structures

If you have used EXPAND or NO-EXPAND on the corresponding DB2 CREATE command for this member, you should use the same keywords for this command, by entering either:

```
DB2 SIZE member EXPAND ;
```

or

```
DB2 SIZE member NO-EXPAND ;
```

where *member* is the name of a DB2-TABLE or DB2-INDEX member. EXPAND attributes in the DB2-TABLE member are used as the default.

Refer to "DB2 CREATE" on page 212 for use of the EXPAND/NO-EXPAND keywords.

Refer to the DB2-TABLE member type in Chapter 9, "Repository Member Types," on page 341 for details of the EXPAND attribute.

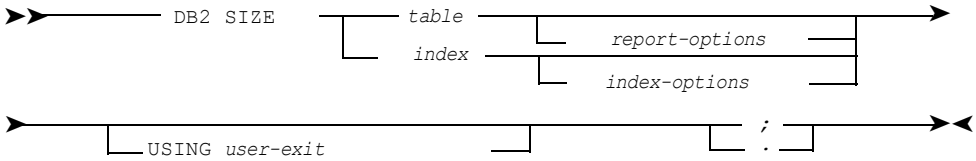
Time, Date, and Timestamp Usage - Field Allocations

Time, Date, and Timestamp usage fields are allocated (fixed) internal lengths of 3, 4, and 10 bytes respectively.

NON-UNIQUE/UNIQUE - Index Size Calculations

Calculations will be made for an index as NON-UNIQUE provided that the member does not have UNIQUE specified in its definition. This condition can also be simulated for a member with the UNIQUE keyword, by invoking the DB2 RECALCULATE command with a DUPLICATES value greater than 1.

DB2 SIZE Syntax



where:

table is the name of a DB2-TABLE repository member.

report-options are:

index is the name of a DB2-INDEX repository member

index-options are:

```
──┬──────────────────┬──┬──────────────┬──────────┬──────┬──┬──────────────
  │ report-options   │  │ NOT-NULL     │ ALL      │      │  │
  └──────────────────┘  │              │ NONE     │      │  │
                        │              │ REFER    │      │  │
                        └──────────────┴──────────┴──────┴──┘
```

where:

report-options are as defined above.

user-exit is the name of an executive routine.

DB2 SYNONYM

DB2 SYNONYM generates SQL CREATE SYNONYM or DROP SYNONYM statements.

Refer to "DB2 SYNONYM Syntax" on page 297 for the syntax of the DB2 SYNONYM command.

To generate SQL CREATE SYNONYM or DROP SYNONYM statements from a DB2-USER repository member, enter either:

```
DB2 SYNONYM CREATE user ;
```

or

```
DB2 SYNONYM DROP user ;
```

where *user* is the name of a DB2-USER repository member.

To generate other SQL CREATE or DROP statements, use the DB2 CREATE or DB2 DROP command.

You can generate SQL statements for a table or a view. A separate SQL statement is generated for each DB2-TABLE or DB2-VIEW member named in the SYNONYMS attribute of the DB2-USER member. You can also generate the last synonyms in a DB2-USER member, for example, if new synonyms have been added to an existing list and you only need to generate the additions.

When you have applied an SQL DROP SYNONYM statement to your DB2 environment, you should remove or update the relevant DB2-USER member to reflect the changes.

Generated output is displayed on the screen. You can tailor this output, file it on the MP-AID, or send it to an external dataset, using output generation options. You can also tailor output by calling executive routines (*user exit routines*) at set points (*user exits*) during output. This process is known as taking user exits.

The systems administrator can tailor output using the DB2 profile.

Refer to"Tailoring Output" on page 109 for details of the DB2 profile and user exits.

Refer to the DB2-USER member type in Chapter 9, "Repository Member Types," on page 341 for an example of an SQL CREATE SYNONYM statement generated by the DB2 SYNONYM command.

Name Editing Options

Use the REPLACE/REPLACING, INSERTING, and DROPPING keywords to edit generated data names before they are output.

Refer to "Name Editing Options" on page 337 for further details of name editing options.

Taking User Exits

To take a user exit, enter either:

```
DB2 SYNONYM CREATE user USING exit-routine ;
```

or

```
DB2 SYNONYM DROP user USING exit-routine ;
```

where *exit-routine* is the name of an executive routine.

Note:

The systems administrator can alter your DB2 Profile so that a default user exit is always taken with this command. The USING keyword will override any default user exits set this way.

Refer to "Tailoring Output" on page 109 for further details of user exits.

Specifying an Owner

To select a DB2-USER member with a specified owner, overriding any owner defined for that member, enter:

```
DB2 SYNONYM CREATE user SQLID owner ;
```

or

```
DB2 SYNONYM DROP user SQLID owner ;
```

where *owner* is a delimited string of up to 8 characters, giving the authorization ID of a specific user.

Output Generation Options

Use the ONTO keyword to direct your generated SQL statements to a specific destination. This destination can be:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset.

Refer to "Output Generation Options" on page 334 for further details of output generation options.

Generating the Last Synonyms in a DB2-USER Member

To generate only the last synonyms listed in a DB2-USER member, enter either:

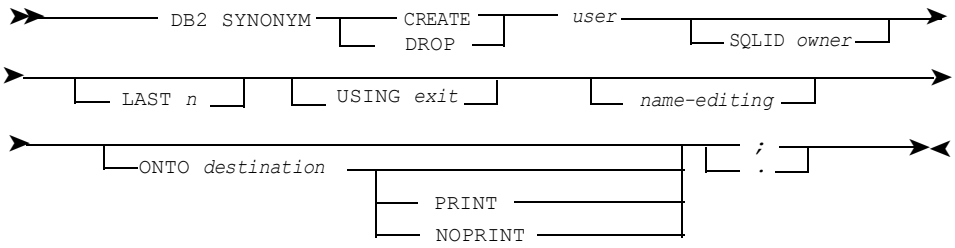
```
DB2 SYNONYM CREATE user LAST n ;
```

or

```
DB2 SYNONYM DROP user LAST n ;
```

where *n* is a number, specifying how many synonyms (taken from the end of the list in the DB2-USER member) are to be selected.

DB2 SYNONYM Syntax



where:

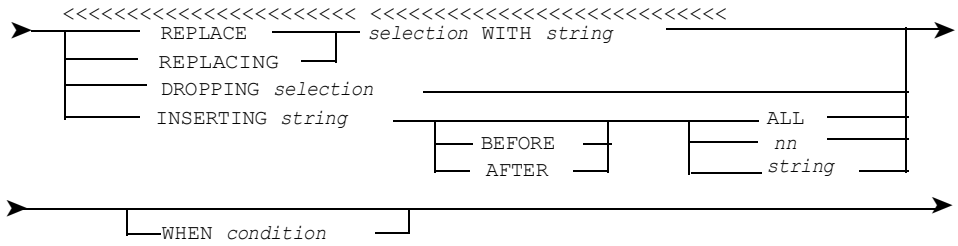
user is the name of a DB2-USER member.

owner is a delimited string of up to 8 characters, giving the authorization ID of a specific user.

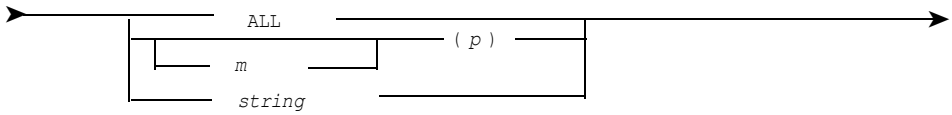
n is an integer specifying how many synonyms are to be selected.

exit is the name of an executive routine.

name-editing is:



where *selection* is:



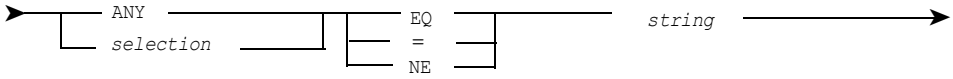
where:

m and *p* are integers in the range 1 to 96.

string is a delimited string of not more than 32 printable characters.

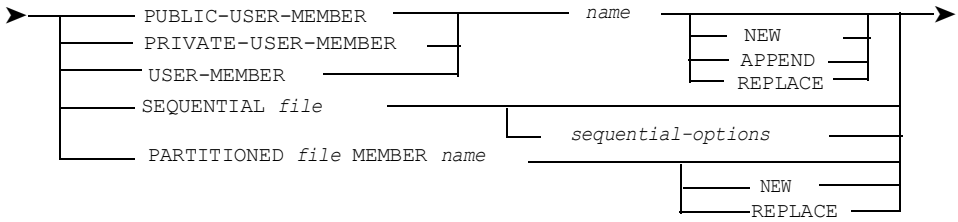
nn is an unsigned integer in the range 1 to 96.

condition is:



where *selection* and *string* are as defined above.

destination is:

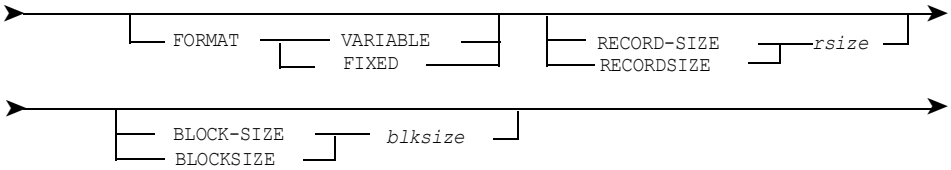


where:

name is the name of a USER-MEMBER.

file is the name of a sequential or partitioned dataset.

sequential-options are:



where:

rsize is the record length

blksize is the block size.

EXTRACT DB2

EXTRACT DB2 imports information about DB2 objects from your DB2 environment.

Refer to "EXTRACT DB2 Syntax2.5" on page 305 for the syntax of the EXTRACT command.

Use the EXTRACT DB2 command to import information about DB2 objects, and store this information in Procedures Language global variables on the WBTA (WorkBench Translation Area).

You can import information about these types of object:

- columns
- databases
- indexes
- owners
- packages
- plans
- storage groups
- tables
- table spaces
- types
- views

You can select objects by their name and owner. If you import information about an object, information on other objects directly related to it is also imported.

You can then use the imported information to generate repository members which document the DB2 objects. The member's definitions will include attributes documenting the relationships of the objects to one another.

If the definition of an object refers to another object on which information is not imported, then the referenced object is added to the repository as a dummy member, if the referenced object is not already in the repository. Examples of referenced objects are DBRMs, and field, edit, and validation procedures.

Refer to *ASG-Manager Products Procedures Language* for further details of global variables.

Ownership of a parent object is the same as that of its children. For example, if you import information about a DB2 table you can document the table, and its columns and ownership in the repository. Tables with which it has referential constraints would be referenced objects.

Refer to the *ASG-DictionaryManager User's Guide* for further details of parent, children, and referenced objects.

Importing Information About DB2 Objects

To import information about a selection of DB2 objects into the Manager Products environment, enter:

```
EXTRACT DB2 object-type object-name-selection ;
```

where:

`object-type` is one of the keywords DATABASE, INDEX, PACKAGE, PLAN, STOGROUP, TABLE, TBSPACE, VIEW, TYPE, PROCEDURE, FUNCTION, or TRIGGER. It identifies the type of object.

`object-name-selection` identifies the names of the objects and is either:

- The name of one or more objects separated by commas
- A combination of characters and question mark (?) and asterisk (*) symbols which together match the names of a selection of objects

Each object name must be the name (excluding any owner qualifier) of the object as it is known on the DB2 catalog. The owner ID of each object defaults to your Manager Products Logon Identifier.

If your Manager Products Logon Identifier is different than the authorization ID of the owner of the DB2 object you can only import information about it if you specify the correct authorization ID in the CREATOR keyword of the EXTRACT DB2 command.

To import information about DB2 tables stored in specific tablespaces, enter:

```
EXTRACT DB2 object-type object-name-selection TBSPACE
object-type object-name-selection;
```

Importing Information About Objects Owned by any Owner

To import information about objects owned by particular owners, enter:

```
EXTRACT DB2 object-type object-name-selection CREATOR
owner-selection ;
```

where *owner-selection* identifies the authorization IDs of the owners of the objects and is either:

- The authorization IDs of one or more owners separated by commas
- A combination of characters and question mark (?) and asterisk (*) symbols which together match the authorization IDs of a selection of owners

Selecting Object Names and Owner Authorization IDs

With one EXTRACT command you can import information about one or more objects owned by one or more owners, using the question mark (?) and asterisk (*) symbols to match the names of objects and the authorization IDs of the owners of those objects.

The question mark (?) represents any single character. For example, if you specify an object name of EMP-REC and an authorization ID of ???? in an EXTRACT command then all objects named EMP-REC owned by any owner with a four-character authorization ID are selected.

The asterisk (*) symbol represents any string of zero or more characters. For example, if you specify an object name of * and an authorization ID of DJB in an EXTRACT command then all objects owned by the user with the authorization ID DJB are selected.

You can combine question mark (?) and asterisk (*) symbols together and with other characters. For example, if you specify an object name of TAB-* and an authorization ID of * in an EXTRACT command, then objects are selected, owned by any owner, that have names (of any character length) staring with TAB-.

Note:

You cannot represent object names or authorization IDs with question mark (?) and asterisk (*) symbols in the same keyword of an EXTRACT command in which you have also specified a list of object names or authorization IDs separated by commas.

Expanding Packagelist Entries when Importing Plans

You may bind a plan with packagelist entries that contain an asterisk (*) to indicate ALL in any of the entry's LOCATION, COLLECTION, or PACKAGE columns.

When you import such a plan, you may specify whether any collection or package columns in the pklist entry containing an asterisk (*) are expanded. If the LOCATION specified is remote, however, you may not expand the pklist entry, because the necessary information regarding the collections and packages at that location is not available. If the LOCATION column contains an asterisk (*), the requested details are extracted from the local location only.

Command variable mpdy_expand_pklist, defined in exec MPDY42DFLT, controls expansion. If expansion of the pklist entries is required, this variable should be set to :Y:, its default setting.

For example, if SYSIBM.SYSPKLIST contains an entry *.*.PK1 where PLANNAME is the plan being extracted and package PK1 is bound in collections COLL1, COLL2, COLL3, these members would be generated:

- PK-COLLI-PK1
- PK-COLL2-PK1
- PK-COLL3-PK1
- CL-COLL1
- CL-COLL2
- CL-COLL3.

If only one DB2-PACKAGE member is to be generated regardless of any multiple selections in the list, then MPDY_EXPAND_PKLIST should be set to :N:. The naming standards are then as seen in Table 21.

Packages on local subsystem are: COLL1.PK1

Table 21 Naming Standards

SYSIBM.SYSPKLIST Entry	Members Generated	
REMOTLOC.*.*	PK-RENOTLOC-ALL-ALL CL-REMOTLOC-ALL LO-RENOTLOC	(SOL-ALIAS *) SQL-ALIAS *
..*	PK-ALL-ALL-ALL CL-ALL-ALL LO-ALL	SQL-ALIAS * SQL-ALIAS * SQL-ALIAS *

Table 21 Naming Standards

SYSIBM.SYSPKLIST Entry	Members Generated	
COLL1.*	PK-COLL1-ALL CL-COLL1	SQL-ALIAS *
*.PK1	PK-ALL-PK1 CL-ALL	* SQL-ALIAS *
LOCALLOC.*.*	PK-LOCALLOC-ALL-ALL CL-LOCALLDC-ALL LO-LOCALLOC	SQL-ALIAS * SQL-ALIAS *

When the indicator MPDY_EXPAND_PKLIST = :N:, it is irrelevant whether the location specified is local or remote.

The SQL alias allows generation of PKLIST(*.*.* , *.PK1 ...) during DB2 BIND.

The Information Which Can Be Imported from DB2

Table 22 shows the information which can be imported from DB2:

Table 22 Information Which Can Be Imported from DB2

EXTRACT Command	Objects on which Information is Imported	Repository Member Type
EXTRACT DB2 TABLE	A table and:	DB2-TABLE
	• Its owner	DB2-user
	• Its columns and their field procedures	ITEMs * MODULEs
	• Its validation procedure	* MODULEs
	• Its edit procedure	* MODULEs
	• The table space it is in	* DB2-TBSPACE
	• The tables with which it has referential constraints.	* DB2-TABLEs
EXTRACT DB2 VIEW	A view and:	DB2-VIEW
	• Its owner	DB2-USER
	• The columns that make up the view	* ITEMs
	• The tables upon which the view is based	* DB2-TABLEs
	• The views upon which the view is based.	* DB2-VIEWs

Table 22 Information Which Can Be Imported from DB2

EXTRACT Command	Objects on which Information is Imported	Repository Member Type
EXTRACT DB2 INDEX	An index and:	DB2-INDEX
	• Its owner	DB2-USER
	• The storage group it is in	* DB2-STOGROUP
	• The table it is indexing	* DB2-TABLE
	• The columns forming the index key.	* ITEMs
EXTRACT DB2 PLAN	A plan and:	DB2-PLAN
	• Its owner	DB2-USER
	• Its bound DBRMs	* MODULEs
	• Its bound PACKAGES.	DB2-PACKAGE DB2-LOCATION DB2-COLLECTION
EXTRACT DB2 DATABASE	A database and:	DB2-DATABASE
	• Its owner	DB2-USER
	• The default storage group associated with it.	* DB2-STOGROUP
EXTRACT DB2 TBSPACE	A table space and:	DB2-TBSPACE
	• Its owner	DB2-USE
	• The storage group it is in	DB2-STOGROUP
	• The database it is in.	* DB2-DATABASE
EXTRACT DB2 ALIAS	An alias and:	DB2-ALIAS
	• Its owner	DB2-USER
	• The location the alias refers to.	DB2-LOCATION
EXTRACT DB2 STOGROUP	A storage group and:	DB2-STOGROUP
	• Its owner.	DB2-USER
EXTRACT DB2 PACKAGE	A packagelist entry	DB2-PACKAGE DB2-COLLECTION
EXTRACT DB2 PROCEDURE	A procedure definition	DB2-PROCEDURE
EXTRACT DB2 FUNCTION	A function definition	DB2-PROCEDURE

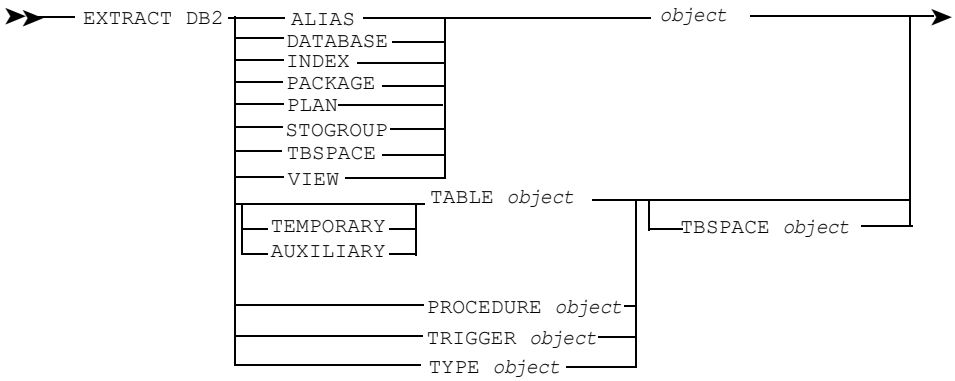
Table 22 Information Which Can Be Imported from DB2

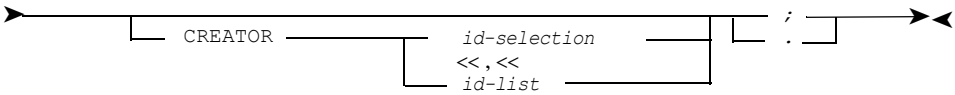
EXTRACT Command	Objects on which Information is Imported	Repository Member Type
EXTRACT DB2 TRIGGER	A trigger.	DB2-TRIGGER
EXTRACT DB2 TYPE	Distinct type.	ITEM

Note:

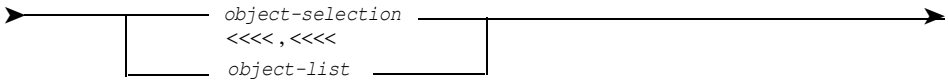
Those member types marked with an asterisk (*) are referenced objects, created as dummy members, if they do not exist already on the repository.

EXTRACT DB2 Syntax





where *object* is:



where:

object-selection is a combination of characters and ? and * symbols, matching the name of a selection of external objects.

object-list is a list of one or more names of objects to be extracted.

where the question mark (?) represents any single character and an asterisk (*) represents a string of (any number of) any characters.

id-selection is a combination of characters and question mark (?) and asterisk (*) symbols matching the authorization IDs of a selection of owners.

id-list is a list of one or more names of IDs to be extracted.

ISQL

ISQL dynamically submits SQL statements to your relational environment.

Refer to "ISQL Syntax" on page 308 for the syntax of the ISQL command.

Submitting SQL Statements

You can use the ISQL command to dynamically submit SQL statements:

- Entered in the Command Area
- Printed in the current buffer
- Filed in a USER-MEMBER

to your relational environment. By default, SQL statements in the current buffer are submitted.

To submit SQL statements entered in the Command Area, enter:

```
ISQL sql-statement ;
```

where *sql-statement* is any SQL statement that can be dynamically prepared for execution. The SQL statement can be up to 255 characters long including leading, embedded, and trailing blanks. SQL SELECT statements must conform to the specifications of a full select statement.

To submit SQL statements printed in the current buffer, enter:

```
ISQL ;
```

The current buffer can be a Command Mode, Edit, Update, or Lookaside Buffer.

To submit SQL statements filed in a USER-MEMBER, enter:

```
ISQL user-member-name ;
```

where *user-member-name* is the name of the USER-MEMBER in which the SQL statements are filed.

You can dynamically submit to your DB2 or SQL/DS environment SQL statements generated by a previous Manager Products DB2 or SQL command. For example, you can generate a CREATE TABLE statement with the DB2 CREATE or SQL CREATE commands. The CREATE TABLE statement can be either printed or filed in a USER-MEMBER.

The ISQL command will attempt to submit the entire contents of the USER-MEMBER or current buffer except for comment lines.

Comment lines preceded by two or more hyphens are displayed by the DB2 or SQL command and describe the SQL statement generated.

If the output generated by the DB2 or SQL command includes Manager Products messages then they are also submitted and may cause the SQL statement to be rejected by DB2 or SQL/DS. SQL statements filed in a USER-MEMBER do not include messages. You can also use the SWITCH MESSAGES command to stop Manager Products messages being displayed in the current buffer.

Refer to the *ASG-ControlManager User's Guide* for details of the SWITCH MESSAGES command.

Restricting the Number of Rows of Result Tables

You can specify the number of rows in a result table to be printed within the Manager Products environment in response to SQL SELECT statements submitted with the ISQL command.

To restrict the size of result tables, enter:

```
ISQL n ;
```

where *n* is the maximum number of rows to be printed. The first *n* rows in a result table are printed.

The size restriction applies to all result tables you generate with the ISQL command for the rest of the current Manager Products session.

You can change the maximum size of result tables by entering another ISQL command specifying an alternative number of rows.

The size restriction does not apply to result tables printed by SELECT statements submitted dynamically to your relational environment from within executive routines. You can specify the number of rows to be printed by an executive routine by including an SQLI_ROWS variable in the executive routine.

Refer to "Control Variables" on page 139 for details of the SQLI_ROWS variable.

Querying SQL/DS SQLCODEs

You can display within your Manager Products environment SQL/DS HELP text explaining SQLCODEs.

To display SQL/DS HELP text, enter either:

```
ISQL HELP sql/ds-sqlcode ;
```

or

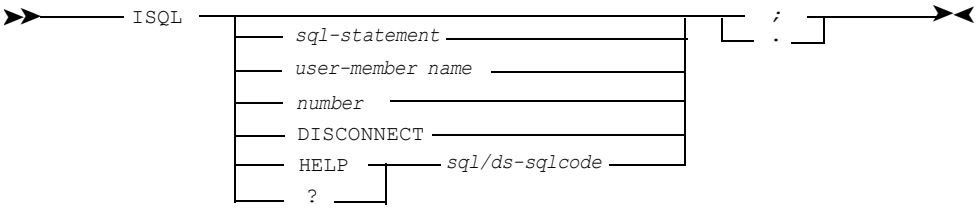
```
ISQL ? sql/ds-sqlcode ;
```

where *sql/ds-sqlcode* is an SQL/DS SQLCODE number.

SQL/DS SQLCODEs and HELP text are always displayed in response to unsuccessful SQL statements submitted to your SQL/DS environment with the ISQL command.

Refer to "Output" on page 133 for details of the output of the ISQL command.

ISQL Syntax



where:

sql-statement is any SQL statement that can be dynamically prepared for execution. The SQL statement can be a maximum of 255 characters long including leading, embedded and trailing blanks. SQL SELECT statements must conform to the specifications of a full select statement.

user-member-name is the name of a USER-MEMBER in which an SQL statement is filed.

DISCONNECT disconnects the user from the current DB2 subsystem specified in MPDY42DFLT.

sql/ds-sqlcode is an SQL/DS SQLCODE number.

number is the number of rows in a result table to be printed in response to a SQL SELECT statement.

POPULATE

The POPULATE command populates the repository with the results of the preceding PREVIEW IMPORT command.

Refer to "POPULATE Syntax" on page 310 for the syntax of the POPULATE command.

Populating the Repository

Use the POPULATE command to execute the ADD or REPLACE command and member definition statements generated by a previous PREVIEW IMPORT command. The statements are those displayed in the current buffer or filed in a USER-MEMBER.

To execute statements displayed in the current buffer, enter:

```
POPULATE FROM BUFFER ;
```

The output of the PREVIEW command must be displayed in the current buffer which can be a Command Mode, Lookaside, Update, or Edit Buffer.

If you want to enter other commands between a PREVIEW and POPULATE command you can prefix these other commands with BROWSE or LOOKASIDE and then use QUIT or XQUIT, to return to the output of the PREVIEW command, before entering POPULATE.

To execute ADD and REPLACE statements filed in a USER-MEMBER enter:

```
POPULATE FROM USER user-member-name ;
```

where *user-member-name* is the name of the USER-MEMBER in which the command and member definition statements generated by a previous PREVIEW IMPORT command have been filed.

You can also execute the command and member definition statements by editing the USER-MEMBER and entering a RUN command or by entering the name of the USER-MEMBER in the Command Area.

By prefixing POPULATE with a NOPRINT command you can stop any output being printed.

Refer to the *ASG-ControlManager User's Guide* for details of the commands mentioned above.

Specifying that Statements Will Form a Logical Unit of Work

You can specify that the command and member definition statements entered in the repository by the POPULATE command are to be treated as one Logical Unit of Work (LUW).

To specify that all the statements will form one LUW, enter:

```
POPULATE FROM BUFFER ROLLBACK ;
```

or

```
POPULATE FROM USER user-member-name ROLLBACK ;
```

By using the ROLLBACK keyword you can specify that all the statements will form a LUW which will either update the repository or be rolled back in its entirety, leaving the repository unchanged, if for any reason any of the statements are unsuccessful.

For example, you can avoid a situation where the definition of the parent object is entered in the repository but the definitions of some of its children are not.

To change the message error level which causes ROLLBACK to occur, enter:

```
POPULATE FROM BUFFER ROLLBACK LEVEL error-level ;
```

or

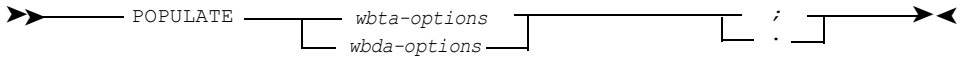
```
POPULATE FROM USER user-member-name ROLLBACK LEVEL error-level ;
```

where *error-level* can be:

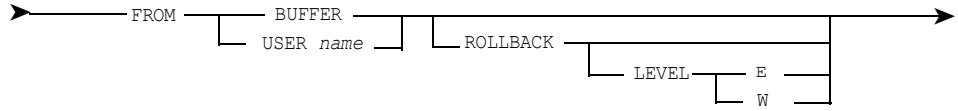
- E, to cause ROLLBACK when Error messages occur
- W, to cause ROLLBACK when Warning messages and Error messages occur.

If you do not use the LEVEL keyword, E is the default (so ROLLBACK is not normally caused when Warning messages occur).

POPULATE Syntax



where *wbta-options* are:



where *name* is the name of a USER-MEMBER on the MP-AID.

wbda-options are:

```
➤──┬─ ENTITIES ──┬─┬─ PREFIX-USERVIEWS ──┬─ string ─┬──┬──➤
   │             │ │                     │          │  │
   │             │ └─ SUFFIX-USERVIEWS ──┘          │  │
   └─ USERVIEWS ─┴──────────────────────────────────┴──┘

➤──┬─ AS-MODEL viewset-name ─┬──┬─ NAMES name-list ──┬──➤
   │                         │  │                    │
   │                         │  └─ NUMBERS range-list┘
   └─────────────────────────┘

➤──┬─ ALPHABETICALLY ─┬──┬─ USING FORMAT format ─┬──➤
   │                  │  │                       │
   └──────────────────┘  └───────────────────────┘
```

where:

string and *viewset-name* can contain from 1 to 32 characters and must conform to the Manager Products rules for valid member names.

Note:

If a supplementary USERVIEW member is required for an ENTITY being defined, string is concatenated, either as a prefix or a suffix, with the name of the ENTITY, to produce the name of the supplementary USERVIEW. If this resulting name contains more than 32 characters, it is reduced.

name-list is a list of names, separated by commas, of relations or records, depending, respectively, on whether USERVIEWS or ENTITIES has been specified in the command

range-list is:

```
      <<<<<< , <<<<<<<<<<<<<<<< <<<<<<<<<
➤──── m ──┬──────────────────────────────────────────────➤
          │
          └──── TO n ────┘
```

where *m* and *n* are numbers assigned in the Workbench Design Area to relations or records, depending respectively, on whether USERVIEWS or ENTITIES has been specified in the command. If present, *n* must be greater than *m*.

format is the name of a FORMAT member of the Modeling Repository. FORMAT members are used by User Formatted Output functions.

Note:

wbta-options are provided by generic import functions.
wbda-options are provided by data modeling and design functions.

PREVIEW IMPORT

PREVIEW IMPORT generates ADD or REPLACE command and member definition statements from the information on the WorkBench Translation Area (WBTA) documenting external objects.

Refer to "PREVIEW Syntax" on page 314 for the syntax of the PREVIEW command.

The PREVIEW IMPORT command uses the information on the WorkBench Translation Area (WBTA) as it has been processed by previous RECONCILE, RADD, RIGN, RREN, RREP, or RUPD commands to generate ADD or REPLACE command and member definition statements.

To generate the command and member definition statements, enter:

```
PREVIEW IMPORT ;
```

The member definition statements are generated in the default layouts provided by Manager Products for each member type. You can create layout rules that suit your repository standards and invoke them in the USING keyword of the PREVIEW IMPORT command.

The NOTE attribute contains the date and time that the member definition statement was generated by the PREVIEW command. The ALIAS attribute contains the external object's name. The alias type will correspond to the language used in the external environment from which information about the object was imported.

Information in the NOTE and ALIAS attributes of the existing member is incorporated in those of the proposed member. The single ALIAS attribute generated could contain different aliases of the same alias type. You must edit the member definition statement generated by the PREVIEW command and change one of the aliases.

If the definition is to replace an existing member, then certain default common attributes of the existing member are incorporated in the definition unless you have specified the NO-COMMON-CLAUSES keyword in a previous RECONCILE command.

PREVIEW IMPORT processes members on the WBTA in the same order as they are listed on the Reconciliation Report generated by the previous RECONCILE command. A proposed member can appear more than once in a Reconciliation Report but the PREVIEW IMPORT command only generates one command and member definition statement for each member.

For example, you could import information about two tables which share a column of the same name. The shared column would appear twice on the Reconciliation Report but only one command and member definition statement would be generated to document it in the repository.

A member whose definition is not generated, because:

- It has been ignored by a previous RECONCILE command
- It has already been generated in the current PREVIEW IMPORT output

is indicated by comments. These comments help you to relate the PREVIEW output with the previous Reconciliation Report.

The generated statements can be:

- Printed
- Filed in a USER-MEMBER on the MP-AID
- Both printed and filed.

To file the generated statements in a USER-MEMBER you must specify an ONTO keyword in the PREVIEW IMPORT command.

By filing the command and member definition statements in a USER-MEMBER you can:

- Hold them across Manager Products sessions
- Edit the generated statements in the Edit Buffer.

You can subsequently enter the statements in the repository using the POPULATE command.

Generating Member Definition Statements in Your Own Layouts

You can tailor the PREVIEW IMPORT command so that it generates member definition statements in layouts which suit your repository standards.

To generate member definition statements in your own layouts enter:

```
PREVIEW IMPORT USING layout-executive ;
```

where *`layout-executive`* is an executive routine which invokes other executive routines which each determine the layout of member definition statements generated for a particular member type.

Alternatively you can tailor the executive routines in Manager Products default layout rules.

Refer to "Tailoring Import" on page 157 for further details of tailoring the PREVIEW IMPORT command.

Filing Generated Output in a USER-MEMBER

To automatically file generated output in private or public USER-MEMBERs on the MP-AID, enter:

```
PREVIEW IMPORT ONTO member-type member-name options ;
```

where:

member-type is the type of USER-MEMBER which will hold the generated output: USER-MEMBER, PUBLIC-USER-MEMBER, or PRIVATE-USER-MEMBER, to file output in a public or private USER-MEMBER (USER-MEMBER is private).

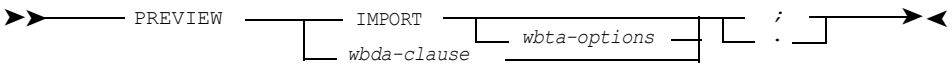
When appending or replacing the contents of an existing member, the user who created that member can change it from private to public, or the reverse, by specifying either PRIVATE or PUBLIC. Users with different Logon Identifiers can create private USER-MEMBERs with the same names.

member-name is the name of the USER-MEMBER.

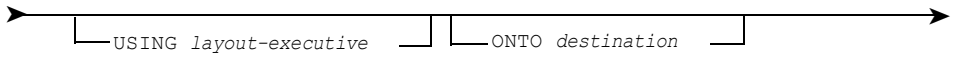
options define how the output is to be filed. These keywords are as follows:

- NEW (the default), APPEND, or REPLACE, to create a new member, append to an existing member, or replace an existing member. If you specify NEW, and the member already exists, the output will not be generated. If you specify APPEND or REPLACE, and the member does not already exist, a new member is created.
- PRINT (the default), to print the full output, or NOPRINT, to print messages and impact analysis reports only.

PREVIEW Syntax



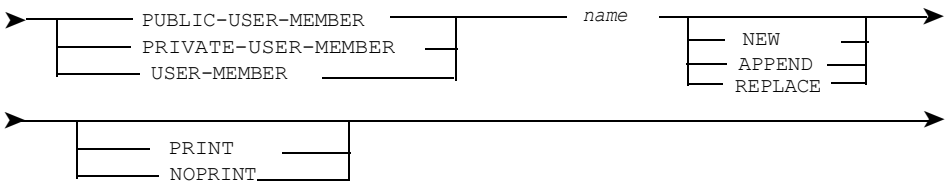
where *wbta-options* are:



where:

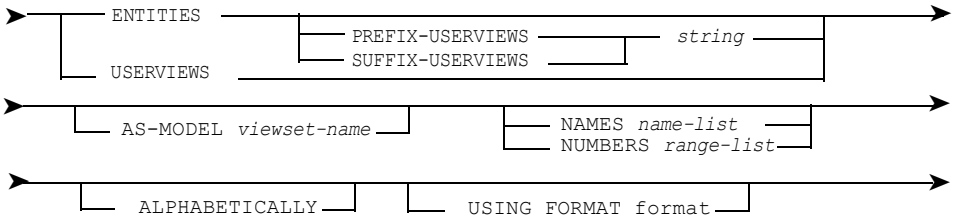
layout-executive is the name of an executive routine

destination is:



where *name* is the name of a USER-MEMBER on the MP-AID.

wbda-clause is:



where:

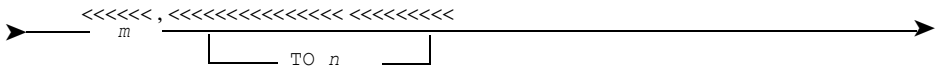
string and *viewset-name* can contain from 1 to 32 alphanumeric characters and must conform to the Manager Products rules for valid member names.

Note:

If a supplementary USERVIEW member is required for an ENTITY being defined, string is concatenated, either as a prefix or a suffix, with the name of the ENTITY, to produce the name of the supplementary USERVIEW. If this resulting name contains more than 32 characters, it is truncated.

name-list is a list of names, separated by commas, of relations or records, depending, respectively, on whether USERVIEWS or ENTITIES has been specified in the command

range-list is:



where *m* and *n* are numbers assigned in the WorkBench Design Area to relations or records, depending, respectively, on whether USERVIEWS or ENTITIES has been specified in the command. If present, *n* must be greater than *m*.

format is the name of a FORMAT member of the Modeling Repository. FORMAT members are used by User Formatted Output functions.

Note:

The PREVIEW IMPORT command is provided by generic import functions.
The *wbda-clause* is provided by data modeling and design functions.

RADD

RADD specifies you want a proposed member documenting an external object to be added to the repository.

To use the RADD Line Command, enter:

```
RADD
```

in the Line Command Area alongside the proposed member's identification number in the Reconciliation Report.

To use the RADD Primary Command, enter:

```
RADD n ;
```

in the Command Area.

where *n* is the proposed member's identification number in the Reconciliation Report.

The Reconciliation Report is displayed by the RECONCILE command. To display the changes you have made with the RADD command enter a further RECONCILE command.

The effects of the two forms of the RADD command are the same but you can only enter Line Commands when working in an interactive environment. You can enter several RADD Line Commands at the same time.

If a member with the same name as a proposed member already exists in the repository and you specify RADD, it will be taken to mean replace.

You can also specify that you want a proposed member to be added to the repository by including an ADDING keyword in the RECONCILE command.

Refer to the RECONCILE command for full details of adding proposed members.

RADD Syntax (Line Command)

```
>>──── RADD ──────────────────────────────────────────────────><
```

RADD Syntax (Primary Command)

```
>>──── RADD n ──┬──────┬──┬──────────────────────────────────><
                └──────┘  │ ;
                          └──┘
```

where *n* is a proposed member's identification number in a Reconciliation Report.

RECONCILE

RECONCILE generates proposed members from the information about external objects held on the WorkBench Translation Area and reconcile the proposed members with the current contents of the repository.

Refer to "RECONCILE Syntax" on page 328 for the syntax of the RECONCILE command.

The RECONCILE command uses translation rules to generate proposed members from the information about external objects placed on the Workbench Translation Area (WBTA) by the EXTRACT command.

You can override the translation rules by specifying an ADDING, IGNORING, NO-COMMON-CLAUSES, RENAMING, or REPLACING keyword in a RECONCILE command.

You can also tailor the Manager Products default translation rules or create your own translation rules and invoke them in the USING keyword of a RECONCILE command.

A member name and member type are generated for the proposed members. A form-description and (depending on the data type of the column it is documenting) a USAGE attribute are generated for proposed members which have an ITEM member type.

The RECONCILE command does not update the repository but specifics the updates you intend to make. These updates are determined by the current contents of the repository. A proposed member will be added to the repository unless a member (other than a dummy member without a source record) of the same name already exists.

Existing ITEM members will be replaced by proposed members whose definitions contain additional form descriptions. The form descriptions of the existing member are included in the definition of the proposed member.

In all other cases, if a proposed member has the same name as an existing member, it will not be updated.

The RECONCILE command can be entered any number of times. Any Manager Products instruction other than LOGOFF or RELEASE GLOBAL can be entered between RECONCILE commands. If you specify an EXTRACT command between RECONCILE commands then the information on the WBTA is replaced and subsequent RECONCILE commands will apply to the latest and not the previously imported information.

The first RECONCILE command generates the proposed members. Subsequent RECONCILE commands can both change the proposed members and, subject to member type checks, specify whether or not they are to be entered in the repository. You can also regenerate the proposed members by entering a RECONCILE command including an INITIALIZE keyword. The proposed members are regenerated according to the translation rules and any changes you have made to the proposed members previously generated are abandoned. A Reconciliation Report is displayed by each RECONCILE command. The report compares the proposed members with existing repository members which have the same name.

You can use the Reconciliation Report to reconcile the proposed and existing repository members with one another. You can also use the RADD, RIGN, RREN, RREP, and RUPD commands during reconciliation. In an interactive environment, these can be Line commands or Primary commands.

The Reconciliation Report displays the condition of the existing members at the time the proposed members were generated or regenerated. Reconciliation Reports displayed in response to subsequent RECONCILE commands not including an INITIALIZE keyword will display any changes you have made to the proposed members but will not display any changes in the condition of the existing members.

The systems administrator can define member type checks that specify the types of existing members to which all the proposed members documenting a parent object and its children can refer. A check can fail, partially fail, or pass.

If the check fails, the proposed member and all other members in the parent-children set cannot be added to the repository. This condition is indicated by an error message in the Reconciliation Report and you cannot override it with an RADD, RECONCILE ADDING, RREP, or RECONCILE REPLACING command.

If the check partially fails, the proposed member can be added to the repository but a warning message in the Reconciliation Report indicates the partial failure.

If the check passes, then the proposed members can be added to the repository as normal.

For example, your systems administrator could specify that columns in tables should preferably be documented in ITEM members but can be documented in GROUPs. This check will fail if an existing SYSTEM member has the same name as a proposed member documenting a column but will only partially fail if the existing member is a GROUP.

Member type checking enables you to take early action to ensure that proposed members will not fail to encode, due to a reference to an existing member with an invalid member type, when the repository is populated.

When a member type check failure occurs, you can rename the offending proposed member to a name that does not exist in the repository or to an existing member name that does not cause a further check failure. A RECONCILE RENAMING command rechecks all the proposed members in the parent-children set.

The updates to be made to the repository are determined by the contents of the current or next visible status. The Reconciliation Report displays the condition of the existing members in the current or next visible status. If you change statuses, the report will continue to display the members as they are visible from the previous status unless you regenerate the proposed members by specifying an INITIALIZE keyword.

Refer to "Tailoring Import" on page 157 for details of how to define member type checking.

Regenerating Proposed Members

To regenerate the proposed members documenting external objects, enter:

```
RECONCILE INITIALIZE ;
```

The proposed members are regenerated according to the default translation rules. Any changes you have made to the previously generated members (for example, renaming them) are abandoned. The Reconciliation Report will display the current condition of any repository member whose name is the same as a proposed member.

Tailoring How Proposed Members are Generated

To tailor the RECONCILE command so that it generates proposed members which suit your repository standards, enter:

```
RECONCILE INITIALIZE USING translation-rule-name-list ;
```

where *`translation-rule-name-list`* is a list of one or more executive routines each separated by a comma. The executive routines must be listed in the order they are to be executed.

For example, if Manager Products' default naming rules for proposed members do not suit your repository standards you can create executive routines specifying alternative rules. Alternatively you can tailor the executive routines in the Manager Products default translation rules.

Refer to "Tailoring Import" on page 157 for further details of tailoring the RECONCILE command.

Stopping Proposed Members being Entered In the Repository

To ignore a selection of proposed members so that they are not subsequently entered into the repository, enter:

```
RECONCILE IGNORING selection ;
```

where *selection* specifies which of the proposed members are to be ignored. Refer to "Selecting Members to be Ignored, Added, or Replaced" on page 322 for details of selection.

The Reconciliation Report will indicate that a proposed member is to be ignored.

You can also use the RIGN command to ignore proposed members.

You cannot ignore proposed members documenting referenced objects.

Adding Proposed Members

To specify that you want a selection of proposed members to be added to the repository (and not ignored), enter:

```
RECONCILE ADDING selection ;
```

where *selection* specifies which of the proposed members you want to be added to the repository. Refer to"Selecting Members to be Ignored, Added, or Replaced" on page 322 for details of selection.

Existing members with the same name as proposed members will be replaced as a result of a RECONCILE ADDING command.

You can also specify that a proposed member will replace an existing member by entering a RECONCILE REPLACING or RREP command.

You can rename proposed members by entering a RECONCILE RENAMING or RREN command.

If member type checking has been enabled by your systems administrator and you want to replace an existing member with a proposed member, then the member type of the existing member is checked against a set of allowed proposed member types. If the check fails, the command will not be executed.

The Reconciliation Report will indicate that the proposed members will be added to the repository as a new member or replace an existing member.

You can also use the RADD command to specify which proposed members are to be added to the repository.

You cannot specify that proposed members documenting referenced objects are to be added to the repository. Referenced objects are added to the repository as dummy members by a reference from the member documenting the parent object, if these members are not already present on the repository.

Replacing Existing Members with Proposed Members

To specify that you want a selection of proposed members to replace existing repository members, enter:

```
RECONCILE REPLACING selection ;
```

where *selection* specifies which proposed members are to replace existing members. Refer to "Selecting Members to be Ignored, Added, or Replaced" on page 322 for further details of selection.

The Reconciliation Report will indicate which of the proposed members are to replace existing repository members.

You can also use the RREP command to specify which proposed members are to replace existing members.

If member type checking is enabled by your systems administrator, then the member type of the existing member being replaced is checked against a set of allowed proposed member types. If the check fails, the command is not executed.

You cannot specify that proposed members documenting referenced objects are to replace existing repository members. A relationship is created in the repository between the proposed member documenting the parent object and the existing member.

Renaming Proposed Members

To rename a selection of proposed members, enter either:

```
RECONCILE RENAMING MEMBER member-name-1 AS member-name-2 ;
```

or

```
RECONCILE RENAMING NUMBER n AS member-name-2 ;
```

where:

member-name-1 is the current name of the proposed member

member-name-2 is the new name

n is the proposed member's identification number in the Reconciliation Report.

You can rename several proposed members with one RECONCILE command by repeating the MEMBER and NUMBER keywords. The Reconciliation Report will display the new member names of the proposed members.

If a proposed member appears more than once in the same Reconciliation Report then a RECONCILE command including a MEMBER keyword will rename all occurrences of the proposed members in the report.

If *member-name-2* is the same as the name of another proposed member in the Reconciliation Report, then the command will be rejected. If it is the same as the name of an existing member, then the existing member is displayed in the Reconciliation Detailed Report.

If you rename children then the proposed member documenting the parent object will refer to the children by their new names.

You may want to rename a proposed member if:

- It has the same name as an existing member
- Its name does not suit your naming standards
- It has failed or partially failed member type checking.

You can create your own naming rules and invoke them in the USING keyword of a RECONCILE command or tailor the executive routines in the Manager Products supplied naming rules. You can also use the RREN command to rename proposed members.

Selecting Members to be Ignored, Added, or Replaced

You can select which proposed members you want to be ignored, added to the repository or replace existing repository members.

To select proposed members by their member type, enter:

```
RECONCILE update TYPE member-type-list ;
```

where:

update is IGNORING, REPLACING, ADDING, or AMEND (reserved for future use).

member-type-list is a list of member types each separated by a comma.

To select proposed members by their member name, enter:

```
RECONCILE update MEMBER member-name-list ;
```

where *member-name-list* is a list of member names each separated by a comma.

To select proposed members by their identification number in the Reconciliation Report, enter:

```
RECONCILE update NUMBER id-number-list ;
```

where *id-number-list* is a list of identification numbers each separated by a comma.

To select all proposed members, enter:

```
RECONCILE update GROUP ALL ;
```

To select those proposed members which have the same name as an existing repository member, enter:

```
RECONCILE update GROUP DUPLICATES ;
```

The different updates and selections can be combined in a single RECONCILE command. For example, to specify that:

- Proposed members will replace existing repository members which have the same member name
- Proposed members with a member type of MODULE and the proposed member with the member name IT-DEPT-NAME will not be entered in the repository

enter:

```
RECONCILE REPLACING GROUP DUPLICATES IGNORING TYPE MODULE
                                                MEMBER IT-DEPT-NAME ;
```

Excluding Common Clauses from the Definition of Proposed Members

To stop the default common attributes of existing repository members being incorporated in proposed members which are replacing them, enter:

```
RECONCILE NO-COMMON-CLAUSES ;
```

You can specify with a RECONCILE REPLACING or RREP command that a proposed member is to replace an existing repository member.

If you do not enter a RECONCILE command including the NO-COMMON-CLAUSES keyword, then the ADMINISTRATIVE-DATA, ALIAS, COMMENT, DESCRIPTION, and NOTE default common attributes of the existing repository member are incorporated in the definition of the proposed member replacing it.

Proposed members always have a NOTE and an ALIAS attribute. The attributes are displayed in the member definition statements generated for the proposed members by a subsequent PREVIEW command. The NOTE attribute gives the time and date the statement was generated. The ALIAS attribute gives the name of the external object the definition is documenting.

If the default common attributes of the existing member are updated after the proposed members were last generated by the RECONCILE command, then the updates are not reflected in the member definition statements generated by the PREVIEW command.

Specifying the Type of Reconciliation Report you want Displayed

You can specify the type of Reconciliation Report you want to be displayed by a RECONCILE command.

To display a summarized Reconciliation Report, enter:

```
RECONCILE LIST SUMMARY ;
```

To display a detailed Reconciliation Report, enter:

```
RECONCILE LIST DETAILS ;
```

To display both a detailed and a summarized Reconciliation Report, enter:

```
RECONCILE LIST BOTH ;
```

The summarized report is displayed by default.

To display a detailed Reconciliation Report excluding certain information about the relationships between objects, enter:

```
RECONCILE LIST DETAILS NO-XREF ;
```

If you specify the NO-XREF keyword then the following are not displayed in the detailed Reconciliation Report:

- The table listing the children of the parent object
- The *Also referred to by* entry which indicates that children are shared by more than one of the parent objects on which information has been imported.

A Description of the Reconciliation Summary Report

The Reconciliation Summary Report lists the proposed members documenting the external objects about which information has been imported.

The ID column contains the unique identification number of the proposed members. The identification number specifies the order in which information about each object was imported. This order is determined by the object's type. The numbering in the report is not in sequence if there is more than one parent object because information on several objects of the same type is imported. Parent objects have the lowest identification numbers and are followed in the report by their children.

The Proposed Member Name column contains the name of the proposed members. The Type column contains the member type of the proposed members.

The Upd column specifies how the repository is to be updated with the proposed members. If ADD is specified, the proposed member is to be added to the repository. If REP is specified the proposed member is to replace an existing member in the repository. If IGN is specified the proposed member is not to be entered in the repository. A "*" symbol indicates that the proposed member is a referenced object and is added to the repository as a dummy member by a reference from the member documenting the parent object (if the member does not already exist).

Initially:

- IGN is specified if there is an existing member with the same name as the proposed member
- REP is specified if the proposed member is an ITEM member with a form description not included in the existing member
- ADD is specified if there is no existing member.

The Condition column shows whether there is an existing repository member with the same name as the proposed member and if it is a dummy, encoded, unverified, or protected member. If the column is blank no member of the same name exists. *NO AUTH is displayed if you do not have the authority to access the existing member. The entries in the Condition column are otherwise the same as those displayed in the Condition column of LIST command output.

If the systems administrator has enabled member type checking and a proposed member fails the check, the error message DM05784E is displayed. If a proposed member partially fails the check, the warning message DM05784W is displayed.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of the LIST command.

An Example of the Reconciliations Summary Report

Figure 35 is an example of a Reconciliation Summary Report:

Figure 35 • Example Reconciliation Summary Report

```
*********************************************************************************************
Reconciliation summary report for extract of table AAW.SALES from DB2.
*********************************************************************************************
ID   Proposed Member Name   Type            Upd       Condition
---------------------------------------------------------------------------------------------
1    TB-AAW-SALES           DB2-TABLE       ADD
2    US-AAW                 DB2-USER        IGN       SCE ENC
3    TS-NORTH               DB2-TBSPACE     *
4    IT-QTY                 ITEM            ADD       *    DUM
5    IT-DESCRIPTION         ITEM            IGN       SCE ENC
6    IT-DELIVERY            ITEM            ADD
7    IT-PRICE               ITEM            REP       SCE ENC
```

A Description of the Reconciliation Detailed Report

The Reconciliation Detailed Report is divided into different sections for each external object about which information has been imported. Each object has an unique identification number which specifies the order in which information about it was imported. This number is the same as the number in the summary report.

The entries following Extracted give the name and type of the external object. The entries following Refers to give the name and type of a referenced object

The entries following Proposed give the member name and member type of the proposed member documenting the external object and indicate whether the member is to be added to the repository, replace an existing dictionary member or be ignored. Proposed members documenting referenced objects are indicated by a "*" symbol.

If a proposed member is an ITEM then its form-description, (depending on the data type of the column it defines) USAGE attribute, and version are also displayed.

The entries following Dictionary give information about the condition (encoded, dummy or unverified) and member type of any repository member with the same name as the proposed member. If the existing member is an ITEM then all versions of its form-description and their associated USAGE attributes are displayed. *NO AUTH is displayed if you do not have the authority to access the existing member.

The section reporting the parent object is followed by a list of the children (including referenced objects) on which information has been imported.

If information has been imported from more than one parent object and they share the same children this is indicated by an *Also referred to by entry* in the sections reporting the shared children.

If the systems administrator has enabled member type checking and a proposed member fails the check, the error message DM05784E is displayed. If a proposed member partially fails the check the warning message DM05784W is displayed.

An Example of the Reconciliation Detailed Report

Figure 36 is an example of a Reconciliation Detailed Report:

Figure 36 • Example Reconciliation Detailed Report

```
*******************************************************************************
Reconciliation detailed report for extract of table AAW.SALES from W2.
*******************************************************************************************

   1   Extracted...   SALES                                   TABLE
       Proposed....   TB-AAW-SALES                            DB2-TABLEADD

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _6 CHILDREN extracted with AAW.SALES_ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
                1   CREATOR
                1   DBSPACE
                4   COLUMNS
   2   Extracted...   AAW                                     CREATOR
       Proposed....US-AAW                DB2-USER             IGN
       Dictionary..SCE ENC               DB2-USER

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   3   Refers to...   NORTH                                   TABLESPACE
       Proposed....   TS_NORTH           DB2-TBSPACE          *

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   4   Extracted...   QTY                                     COLUMN
       Proposed....   IT-QTY             ITEM                 ADD
       Dictionary..   *                  DUM                  ITEM

 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   5   Extracted...   DESCRIPTION                             COLUMN
       Proposed....   IT-DESCRIPTION     ITEM                 IGN
        Form desc..   CHARACTERS 5       VERSION 1
       Dictionary..   SCE ENC            ITEM
        Version 1..   HELD-AS CHARACTERS 5
        Version 2..   ENTERED-AS CHARACTERS 4

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   6   Extracted...   COST                                    COLUMN
       Proposed....   IT-DELIVERY        ITEM                 ADD
        Form desc..   CHARACTERS 10      USAGE DATE           VERSION 1

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   7   Extracted...   PRICE                                   COLUMN
       Proposed....   IT-PRICE           ITEM                 REP
        Form desc..   NUMERIC 6          VERSION 3
       Dictionary..   SCE ENC            ITEM
        Version 1..   HELD-AS CHARACTERS 5
        Version 2..   ENTERED-AS CHARACTERS 4
*******************************************************************************************
```

RECONCILE Syntax

```
>>-- RECONCILE --+-------------------------+-------------------------->
                 |  initialize-clause      |
>--+-------------------------+--+-------------------------+--------->
   |  NO-COMMON-CLAUSES      |  |                <<<<<<<<<<
   |                         |  |  IGNORING selection     |
>--+-------------------------+--+-------------------------+--------->
   |                <<<<<<<<<<  |                <<<<<<<<<<
   |  REPLACING selection    |  |  ADDING selection       |
>--+-------------------------+--+-------------------------+--------->
   |                <<<<<<<<<
```

RIGN

RIGN specifies that you do not want a proposed member documenting an external object to be entered in the repository.

To use the RIGN Line Command, enter:

```
RIGN
```

in the Line Command Area alongside the proposed member's identification number in the Reconciliation Report.

To use the RIGN Primary Command, enter:

```
RIGN n ;
```

in the Command Area.

where *n* is an integer identifying the proposed member's identification number in the Reconciliation Report.

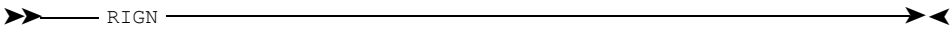
The effects of the two forms of the RIGN command are the same but you can only enter Line Commands when working in an interactive environment. You can enter several RIGN Line Commands at the same time.

The Reconciliation Report is displayed by the RECONCILE command. To display the changes you have made with the RIGN command enter a further RECONCILE command.

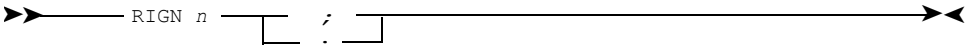
You can also specify that you want a proposed member to be ignored by including an IGNORING keyword in the RECONCILE command.

Refer to the RECONCILE command for full details of ignoring proposed members.

RIGN Syntax (Line Command)



RIGN Syntax (Primary Command)



where *n* is a proposed member's identification number in a Reconciliation Report.

RREN

RREN renames a proposed member during reconciliation.

To use the RREN Line Command, enter:

```
RREN
```

in the Line Command Area alongside the proposed member's identification number in the Reconciliation Report.

To use the RREN Primary Command, enter:

```
RREN n ;
```

in the Command Area.

where *n* is an integer identifying the proposed member's identification number in the Reconciliation Report.

The effects of the two forms of the RREN command are the same but you can only enter Line Commands when working in an interactive environment. You can enter several RREN Line Commands at the same time.

A Dialog Buffer in which you specify the new name of the proposed member is displayed in response to each RREN command.

The Reconciliation Report is displayed by the RECONCILE command. To display the changes you have made with the RREN command enter a further RECONCILE command. You can also rename a proposed member by including a RENAMING keyword in a RECONCILE command.

Note:

This command cannot be performed on a referred object.

Refer to "RECONCILE" on page 317 for full details of renaming proposed members.

RREN Syntax (Line Command)

```
>>---- RREN ----------------------------------------------------><
```

RREN Syntax (Primary Command)

```
>>---- RREN n ----+-----+---------------------------------------><
                  |  ;  |
```

where *n* is a proposed member's identification number in a Reconciliation Report.

RREP

RREP specifies that you want a proposed member documenting an external object to replace an existing repository member.

To use the RREP Line Command, enter:

```
RREP
```

in the Line Command Area alongside the proposed member's identification number in the Reconciliation Report.

To use the RREP Primary Command, enter:

```
RREP n ;
```

in the Command Area.

where *n* is an integer identifying the proposed member's identification number in the Reconciliation Report.

The effects of the two forms of the RREP command are the same but you can only enter Line Commands when working in an interactive environment. You can enter several RREP Line Commands at the same time.

The Reconciliation Report is displayed by the RECONCILE command. To display the changes you have made with the RREP command enter a further RECONCILE command.

You can also specify that you want a proposed member to replace an existing repository member by including a REPLACING keyword in the RECONCILE command.

Refer to "RECONCILE" on page 317 for full details of replacing existing members with proposed members.

RREP Syntax (Line Command)

```
>>──── RREP ──────────────────────────────────────────────────><
```

RREP Syntax (Primary Command)

```
>>──── RREP n ──┬──────┬──────────────────────────────────────><
                └──;───┘
```

where *n* is a proposed member's identification number in a Reconciliation Report.

RUPD

RUPD updates an existing repository member from a Reconciliation Report in order to interactively change its source record.

To use the RUPD Line Command, enter:

```
RUPD
```

in the Line Command Area alongside the identification number in the Reconciliation Report of the proposed member with the same name as the existing member.

To use the RUPD Primary Command, enter:

```
RUPD n ;
```

in the Command Area.

where *n* is the identification number in the Reconciliation Report of a proposed member with the same name as an existing member.

The Reconciliation Report is displayed by the RECONCILE command. To display the changes you have made with the RUPD command enter a further RECONCILE command including the INITIALIZE keyword.

The effects of the two forms of the RUPD command are the same. The RUPD command opens a buffer in Update Mode containing a copy of the source record of the selected repository member which you can then update interactively. You can only enter RUPD commands when working in an interactive environment.

If the selected member is an ITEM, you can copy the form-description and USAGE attribute of the proposed member into the Update Buffer. To do this, use the I, F, and P Line Commands in a Command Interface environment, or the A and B Line Commands in a Panel Interface environment.

To enter the contents of the buffer into the repository use the FILE or SFILE commands. To abandon the update without adding the contents to the repository use the QUIT or XQUIT commands.

You can enter several RUPD Line Commands at the same time. The different Update Buffers are stacked. Use the QUERY ACTIVE-BUFFERS command to find out which buffers you have opened. The number of Update Buffers you can stack is determined by the buffer limit set in the repository by the systems administrator. Use the QUERY BUFFER-LIMIT command to find out the buffer limit.

Note:

The Line Commands only copy the form-description and USAGE attribute of the proposed member corresponding to the existing member at the top of the buffer stack.

Having filed or quit the Update Buffer you will go to an Update Buffer lower in the buffer stack or return to the Reconciliation Report.

The current status must be an update status. If the member does not exist in the current status then the RUPD command copies the source record of the member from the next visible status in which it does exist. If you subsequently FILE or SFILE the member it is entered in the current status.

Refer to the *ASG-ControlManager User's Guide* for details of the FILE, SFILE, QUIT, XQUIT, I, F, P, and QUERY commands.

Refer to the *ASG-MethodManager Workstation User's Guide* for details of the X, A, and B line commands.

RUPD Syntax (Line Command)

```
>>──── RUPD ──────────────────────────────────────────────────────>><
```

RUPD Syntax (Primary Command)

```
>>──── RUPD n ──┬──┬──────────────────────────────────────────────>><
                └──┘
```

where *n* is a proposed member's identification number in a Reconciliation Report.

Output Generation Options

You can use output generation options to send generated output to:

- A USER-MEMBER on the MP-AID (public or private)
- A sequential dataset
- A partitioned dataset
- PRINT

This applies to all output produced by export to DB2 commands, except messages or reports from the following commands: DB2 DROP, DB2 RECALCULATE, DB2 SIZE, DB2 PRODUCE.

For example, you can store a generated SQL statement on the MP-AID, then submit it directly to your DB2 environment, using Dynamic SQL functions, or transfer it to an external dataset using the TRANSFER command.

Refer to Chapter 6, "Dynamic Import/Export," on page 131 for details of Dynamic SQL functions.

Refer to the *ASG-DictionaryManager User's Guide* for details of the TRANSFER command.

Sending Generated Output to a USER-MEMBER

To automatically send generated output to private or public USER-MEMBERs on the MP-AID, enter:

```
command ONTO member-type member-name options ;
```

where:

command is either a DB2 export command, or the PREVIEW command.

member-type is the type of USER-MEMBER which will hold the generated output: USER-MEMBER (private), PUBLIC-USER-MEMBER, or PRIVATE-USER-MEMBER, to file output in a public or private USER-MEMBER.

When appending or replacing the contents of an existing member the user who created that member can change it from private to public, or the reverse, by specifying either PRIVATE or PUBLIC. Users with different Logon Identifiers can create private USER-MEMBERs with the same names.

member-name is the name of the USER-MEMBER.

options define how the output is to be filed. These keywords are:

- NEW (the default), APPEND, or REPLACE, to create a new member, append to an existing member, or replace an existing member. If you specify NEW, and the member already exists, the output will not be generated. If you specify APPEND or REPLACE, and the member does not already exist, a new member is created.
- PRINT (the default), to print the full output, or NOPRINT, to print messages and impact analysis reports only.

Sending Generated Output to a Sequential Dataset

You may wish to send your generated output directly to an external file. To send output to a sequential dataset, enter:

```
command ONTO SEQUENTIAL ddname sequential-options ;
```

where:

command is a DB2 export command.

ddname is the data definition name associated with the sequential dataset. The data definition name must be defined in the job control. If the dataset already exists, and no specification is given with the job control statements or the command, the existing dataset is replaced with the new dataset but has the characteristics of the old dataset.

sequential-options consist of the FORMAT FIXED/FORMAT VARIABLE, RECORD-SIZE, and BLOCK-SIZE keywords, and specify the characteristics of the sequential dataset. These must be defined either in the command or in the job control statements defining the sequential dataset. Characteristics defined in the command take precedence over those defined in job control statements.

For fixed length output, short records are right padded as necessary with blanks.

Note:

The form of the command described here is not available in DOS or CICS environments.

For a full definition and explanation of the characteristics of a sequential dataset, refer to the SENDF command in the *ASG-Manager Products Procedures Language* guide.

Sending Generated Output to a Partitioned Dataset

You may wish to send your generated output directly to a member of a partitioned dataset. This partitioned dataset must already exist.

To send data to a new member of a partitioned dataset, enter:

```
command ONTO PARTITIONED ddname MEMBER name ;
```

or

```
command ONTO PARTITIONED ddname MEMBER name NEW ;
```

where:

command is a DB2 export command.

ddname is the data definition name associated with the partitioned dataset. The data definition name must be defined in the job control.

name is the member of the partitioned dataset to which you wish to send the output.

To replace existing data in a partitioned dataset, enter:

```
command ONTO PARTITIONED ddname MEMBER name REPLACE ;
```

Sending Generated Output to PRINT

To send the output generated to PRINT, enter:

```
command ONTO PRINT ;
```

where *command* is a DB2 export command.

Examples of Output Generation Options

To generate an SQL DROP TABLE statement from the DB2-TABLE member TB-DJB-CUST, enter:

```
DB2 DROP TB-DJB-CUST ONTO PRIVATE DROPCUST REPLACE ;
```

The SQL statement is filed onto an existing private USER-MEMBER named DROPCUST, replacing the current contents of that member.

To send a generated SQL ALTER INDEX statement for the DB2-INDEX member IXFFS-ACCS to the fixed format dataset defined by FILE2 (in the job control), enter:

```
DB2 CREATE IX-FFS-ACCS ONTO SEQUENTIAL FILE2 FORMAT
    FIXED RECORD-SIZE 80 BLOCK-SIZE 8000 ;
```

To generate an SQL CREATE TABLE statement for the DB2-TABLE member TB-AAW-EMPS, appending the generated SQL statements to member MEM15 of the partitioned dataset defined by FILE3 (in the job control), enter:

```
DB2 CREATE TB-AAW-EMPS ONTO PARTITIONED FILE3 MEMBER
    MEM15 APPEND ;
```

Name Editing Options

Use name editing options to edit generated data names before they are output.

You can use name editing options to generate SQL statements to make several copies of an object, or several versions of host structures.

Name editing options for generated SQL statements apply to the object being created, and for all its columns, but not to other referenced objects. For example, in a CREATE TABLE statement, name editing options apply to the table being created, and its columns, but not to tables named in referential constraints or to the tablespace named in the IN attribute.

Name editing for host structures are applied to all generated host variable names.

Names can be expanded up to a maximum of 96 characters during editing. If a generated name is too long for a specific language, it is shortened via the Name Reduction Process.

We recommend caution when editing a generated data name, if you will later import information from a generated object back into your repository. If you have edited the name, it is difficult to reconcile the imported data with the existing member. You should therefore have a rigorous set of naming standards.

You can edit generated data names in three ways:

- Replacing all or part of the name with a specified character string
- Dropping all or part of the name
- Inserting a specified character string at a specified position within the name.

Refer to "Dropping or Replacing a Name" on page 338 and "Inserting a Character String Within a Name" on page 338 for further details of these options.

Refer to "Naming Guidelines" on page 152 for details of naming standards.

Refer to Appendix A, "Name Reduction Process," on page 511 for details of the Name Reduction Process.

Dropping or Replacing a Name

To replace some or all of the name with a specified string, enter either:

```
command REPLACING selection WITH string ;
```

To remove some or all of the name, enter:

```
command DROPPING selection ;
```

where:

command is any DB2 export command

selection is a selected part of the name. It can be either:

- ALL, for the whole of the name (in which case, you must then specify a string to be inserted in its place, using the INSERTING keyword)
- A specified number of characters (*p*), starting at a start position *m* (defaulting to 1). *m* plus *p* must be no more than 97. Names are only edited if the start position plus the number of characters is no more than the length of the name
- String, a delimited string of up to 32 printable characters (including space characters), matching a string in the name

string (as defined above) is to be inserted into the name.

Inserting a Character String Within a Name

To insert a specified string either before or after a specified position in the generated data name, enter either:

```
command INSERTING string BEFORE position ;
```

or

```
command INSERTING string AFTER position ;
```

where:

command is any DB2 export command.

string is a delimited string, of up to 32 printable characters (including space characters), to be inserted.

position is one of these:

- ALL, for the start or end of a name
- *n*, a number giving a character position in the name
- A delimited string of up to 32 printable characters, for a position before or after a matching string in the name.

Examples of Name Editing Options

To generate an SQL CREATE TABLE statement for the DB2-TABLE member TB-EMP-JOBS, replacing the 4-character string starting at position 8 with the string BAK, enter:

```
DB2 CREATE TB-EMP-JOBS REPLACING 8 (4) WITH 'BAK' ;
```

This creates a SQL CREATE TABLE statement referring to a DB2 table called TB-BAK-JOBS. All ITEM/GROUP members documenting columns in that table have their names changed similarly, if they have names that are 12 (8 + 4) or more characters long.

To generate a COBOL data structure from the DB2-TABLE member TB-DJB-EMPS, inserting the string -USER1 at the end of generated variable names, enter:

```
DB2 PRODUCE COBOL FROM TB-DJB-USERS INSERTING '-USER1'
    AFTER ALL ;
```

To generate an SQL ALTER INDEX statement, replacing the string DEV with the string PROD, and dropping the string -TEMP, for the DB2-INDEX member IX-DJB-DEV-TEMP, enter:

```
DB2 ALTER IX-DJB-DEV-TEMP REPLACE 'DEV' WITH 'PROD'
    DROPPING '-TEMP' ;
```

9 Repository Member Types

This chapter includes these sections:

Member Type Descriptions 342
DB2-ALIAS 342
DB2-AUXTABLE 347
DB2-COLLECTION. 350
DB2-DATABASE. 351
DB2-DMS 355
DB2-FUNCTION 363
DB2-INDEX 370
DB2-LOCATION 390
DB2-PACKAGE. 392
DB2-PLAN 398
DB2-PRIVILEGE 405
DB2-PROCEDURE 417
DB2-STOGROUP. 422
DB2-TABLE. 425
DB2-TBSPACE 456
DB2-TEMPTABLE 469
DB2-TRIGGER. 472
DB2-USER 476
DB2-VIEW 481

Reusing Existing Member Definitions. 509

Member Type Descriptions

This section describes the member types in which you document the objects, locations, and the security system of a DB2 database. The member types are documented in alphabetical order of member type name.

You can define repository members as follows:

- In interactive environments, using:
  - Diagrams drawn on a programmable workstation
  - Assisted update panels
  - Update buffers.
- In batch environments, using commands such as ADD.

Throughout this chapter the word *enter* is used to cover all methods of defining members.

When you define a member it is checked to ensure that it follows the syntax. This checking happens, for example, when you file an update buffer or when you use the ENCODE command. Throughout this chapter the phrase "on encoding" is used to cover all methods of encoding members.

Refer to the *ASG-ManagerView User's Guide* for details of using diagrams to define members.

Refer to the *ASG-MethodManager Workstation User's Guide* for details of using assisted update panels to define members.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of using batch and interactive commands to define members, and for details of what happens when a member definition is encoded.

Refer to the *ASG-ControlManager User's Guide* for details of entering member definition statements in different environments.

DB2-ALIAS

DB2-ALIAS defines DB2 aliases in the repository.

Refer to "DB2-ALIAS Syntax" on page 346 for the syntax of DB2-ALIAS member definition.

In the DB2 environment, you can create a local DB2 alias for a table or view at a remote location in a distributed network, if you have access privileges to the remote table or view. Local users who do not know the location of the table or view can then access it. DB2 aliases are represented on the repository as DB2-ALIAS members.

To define a DB2 alias, enter:

```
DB2-ALIAS
```

The member definition must begin with this member type identifier.

All other clauses available to define DB2-ALIAS members are optional. However, for the successful generation of SQL statements, you must define specific clauses, as follows:

- For CREATE ALIAS statements define the CREATOR-OWNER clause
- For DROP ALIAS statements define the CREATOR-OWNER clause
- For COMMENT ON statements define the DB2-COMMENT clause
- For LABEL ON statements define the DB2-LABEL clause.

Note:

The DB2-ALIAS member type is not the same as the common clause ALIAS, which enables you to define alternative names for a repository member.

Refer to Chapter 4, "Repository Definition," on page 91 for details of the generation of DB2 alias names.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-ALIAS member definitions using an AS clause.

Defining an Owner

To define the owner of an alias, enter:

```
CREATOR-OWNER user ;
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the alias.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

The CREATOR-OWNER clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Specifying a Table or View

To specify the table or view to which the DB2 alias applies, enter:

```
FOR object
```

where *object* is a member of one of the following types:

- DB2-TABLE
- DB2-VIEW
- SQL-TABLE
- SQL-VIEW

On encoding, the member specified in the FOR clause is checked to ensure that it is one of the above.

For the successful generation of SQL CREATE ALIAS and DROP ALIAS statements the member named in the FOR clause:

- Must be encoded
- Must have a CREATOR-OWNER clause defined, naming a DB2-USER member
- If your DB2 profile is set to three-part name generation, the DB2-USER member must have a LOCATION clause defined.

Defining a Comment on a DB2 Alias

To define a comment for an alias, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is a string of no more than 254 characters, each line of which is within delimiters.

When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space between the last character of each continuing line and its delimiter.

For example, the DB2-ALIAS named MANAGER-NUMBER has an owner of PERSONNEL, and the following comment defined:

```
DB2-COMMENT 'This table contains the Manager number of every '
'manager in each department'
```

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL_MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department'
```

In this example the word "contains" has been split due to the margins set in the DB2 profile.

This clause must be present for the successful generation of SQL COMMENT ON statements.

Defining a Label on a DB2 Alias

To define a label for an alias, enter:

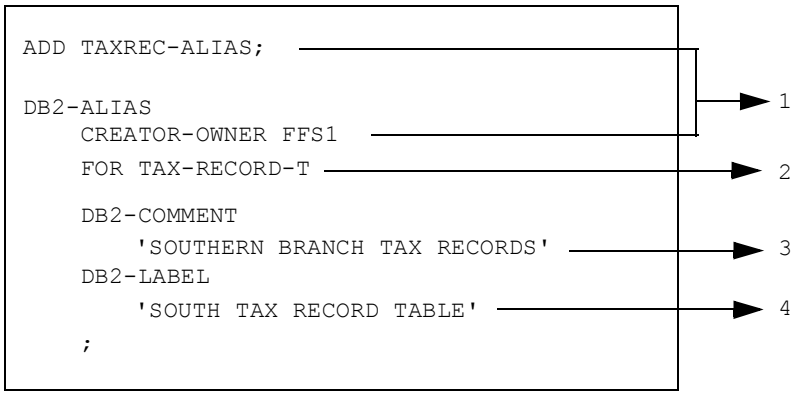
```
DB2-LABEL 'label'
```

where *label* is a string of no more than 30 characters within delimiters.

This clause must be present for the successful generation of SQL LABEL ON statements.

Example

The DB2-ALIAS member TAXPEC-ALIAS is defined in the repository using an ADD command. An SQL statement is generated from the member definition using a DB2 CREATE command.



Repository definition

```
CREATE ALIAS FFS1.TAXREC_ALIAS                    1
   FOR SOUTH.SLJ1.TAX_RECORD_T                    2
;
COMMENT ON ALIAS FFS1.TAXREC_ALIAS
   IS 'SOUTHERN BRANCH TAX RECORDS';              3
;
LABEL ON ALIAS FFS1.TAXREC_ALIAS
   IS 'SOUTH TAX RECORD TABLE';                   4
;
```

SQL Statement

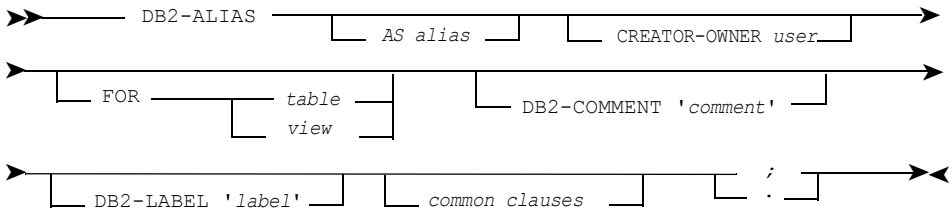
1 The two-part qualified DB2 alias name is taken from the DB2-ALIAS member name and the DB2-USER member, FFS1, named in its CREATOR-OWNER clause. Location-qualifiers are not generated for DB2-alias names.

2 The table name is taken from the DB2-TABLE member TAX-RECORD_T. The DB2 profile is set to three-part name generation, therefore location and owner names are derived and used as qualifiers.

3 The comment and label on the DB2 alias are taken directly from the DB2- ALIAS definition.

4 Same as number 3.

DB2-ALIAS Syntax



where:

alias is the name of a DB2-ALIAS member.

user is the name of a DB2 -USER or SQL-USER member.

table is the name of a DB2-TABLE or SQL-TABLE member.

view is the name of a DB2-VIEW or SQL-VIEW member.

comment is a string of no more than 254 characters, within delimiters.

label is a string of no more than 30 characters, within delimiters.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-AUXTABLE

DB2-AUXTABLE defines DB2 auxiliary tables in the repository. Auxiliary tables are a special subset of the DB2 table, and contains a single column of an LOB type item.

See "DB2-AUXTABLE Syntax" on page 349 for the DB2-AUXTABLE member definition syntax.

Defining an Auxiliary Table

To define a DB2 auxiliary table, enter:

```
DB2-AUXTABLE AUXILIARY
```

Followed by the attributes for this table.

Defining an Owner

To define an owner of this auxiliary table, enter:

```
CREATOR-OWNER user
```

where *user* is a DB2-USER member that identifies the user with the authentication ID for this table. This attribute is required in order to generate an SQL statement using an export command such as DB2 CREATE and DB2 ALTER.

Defining a Comment or Label

To define a DB2 comment, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is a string no longer than 254 characters in length.

To define a DB2 label, enter:

```
DB2-LABEL 'label'
```

where *label* is a string no longer than 30 characters in length.

Specifying Where the Table is Defined

To specify the tablespace where the table is defined, enter:

```
IN tbspace
```

where *tbspace* is the name of a DB2-TBSPACE member which identifies the tablespace.

To specify the database where the table is defined, enter:

```
IN database
```

where *database* is the name of a DB2-DATABASE member which identifies the database. When a database is named, DB2 will generate a tablespace in the named database.

Specifying the Base Table

To specify the base table which contains the column that is to be stored in the auxiliary table, enter:

```
STORES table
```

where *table* is the name of a DB2-TABLE member which identifies the table. If this attribute is used, the CONTAINS attribute must name a column in this table.

Specifying the Form Description

To specify which form description defined in an ITEM member is used to generate the datatype of the column in this table, enter:

```
COLUMNS form-keyword
```

where *form-keyword* is one of these:

- ENTERED-AS
- HELD-AS
- REPORTED-AS
- DEFAULTED-AS

For an auxiliary table, the datatype thus defined must be an LOB type item.

Specifying the ITEM in the table

To specify the ITEM member that defines the column, enter:

```
CONTAINS item
```

where *item* the name of an ITEM member which identifies the LOB type column. Each item can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

where *version* is an integer in the range of 1 to 15, and defines the form description version.

Naming Columns

You can explicitly name the column if you do not want its name to be derived from the ITEM name or its alias. To define the name of the column, enter:

```
KNOWN-AS name
```

where *name* is a string of no more than 18 characters.

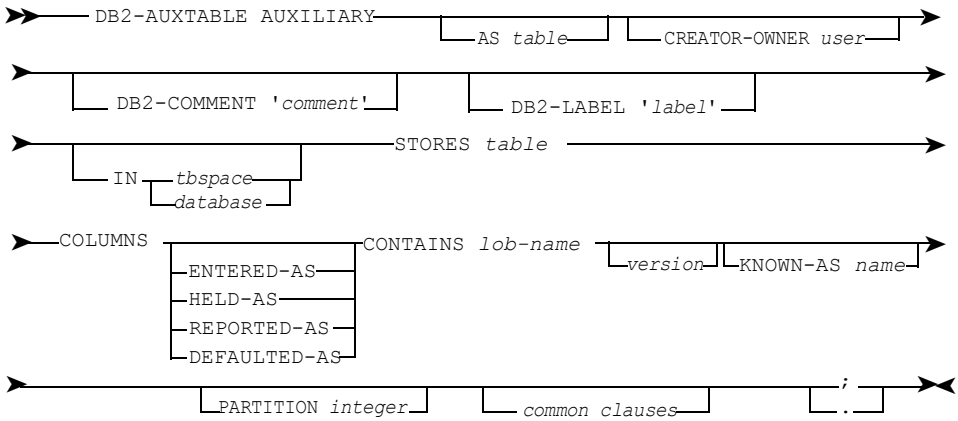
Specifying the Partition

To specify the partition of the base table for which the auxiliary table is to store the specified column, enter:

```
PARTITION integer
```

where *integer* is the partition number in the base table. You can specify PARTITION only if the base table is defined in a partitioned table space, and no other auxiliary table exists for the same LOB column of the base table.

DB2-AUXTABLE Syntax



where:

table is the name of a DB2-TABLE member.

user is the name of a DB2-USER member.

comment is a DB2 comment of up to 254 characters.

label is a DB2 label of up to 30 characters.

tbspace is the name of a DB2-TBSPACE member.

database is the name of a DB2-DATABASE member.

table is as defined above.

lob-name is the name of an LOB type ITEM member.

version is an integer from 1 to 15.

name is the local name of up to 18 characters.

integer is the partition number, from 1 to 64.

DB2-COLLECTION

DB2-COLLECTION defines DB2 collections in the repository.

Refer to "DB2-COLLECTION Syntax" on page 351 for the syntax of the DB2-COLLECTION member definition.

To define a DB2 collection, enter:

```
DB2-COLLECTION
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-COLLECTION members are optional.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-COLLECTION member definitions using an AS clause.

Defining a Location

To define a location for the DB2-COLLECTION enter:

```
LOCATION location-name
```

where *location-name* is the name of a DB2-LOCATION member.

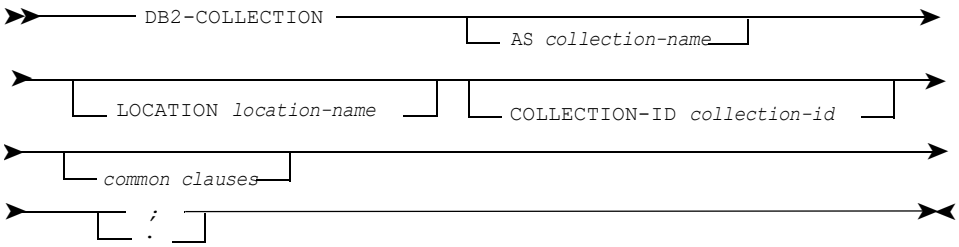
Defining a Collection Identity

To define a collection identity, enter:

```
COLLECTION-ID collection-id
```

where *collection-id* is an identifier of 1 to 18 characters. It is used in the generation of BIND and REBIND utility statements.

DB2-COLLECTION Syntax



where:

collection-name is the name of a DB2-COLLECTION member.

location-name is the name of a DB2-LOCATION member.

collection-id is an identifier of length 1 to 18 characters which must be in delimiters.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-DATABASE

DB2-DATABASE defines DB2 databases in the repository.

Refer to "DB2-DATABASE Syntax" on page 355 for the syntax of the DB2-DATABASE member definition.

To define a DB2 database, enter:

```
DB2-DATABASE
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-DATABASE members are optional.

You can generate SQL CREATE DATABASE and DROP DATABASE statements from DB2-DATABASE members.

Refer to Chapter 4, "Repository Definition," on page 91 for details of the generation of DB2 database names.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-DATABASE member definitions using an AS clause.

Defining a Location

To define the location to which a database belongs, enter:

```
LOCATION location-name
```

where *location-name* is the name of a DB2-LOCATION member that represents a local or remote location in a distributed network.

In DB2-DATABASE member definitions, the LOCATION clause is available to document your DB2 environment but is not used to generate location-qualified names.

Defining a Storage Group

To define the storage group to which the database belongs, enter:

```
STOGROUP stogroup-name
```

where *stogroup-name* is the name of a DB2-STOGROUP member.

On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.

The storage group defined in the DB2-DATABASE definition is the default used by generated table spaces and indexes:

- That belong to the database
- That do not have a storage group specified in their own member definition.

If there is no storage group defined in the DB2-DATABASE member, none is generated and the DB2 default, SYSDEFLT, applies. However you are recommended to explicitly define the storage group, even if it is SYSDEFLT, so that the repository accurately reflects your DB2 environment.

Defining an Associated Buffer Pool

To define the buffer pool that a database uses, enter:

```
BUFFERPOOL bufferpool-name
```

where *bufferpool-name* is one of the following buffer pools: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BP10, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

The buffer pool defined in the DB2-DATABASE definition is the default used by those table spaces and indexes that belong to the database, and do not have a buffer pool specified in their own member definition.

If you do not define a buffer pool in a DB2-DATABASE, none is generated and the DB2 default applies.

Defining the Encoding Scheme

To specify that the data is to be encoded using the ASCII CCSID specified during installation, enter:

```
CCSID ASCII
```

To specify that the data is to be encoded using EBCDIC CCSID specified during installation, enter:

```
CCSID EBCDIC
```

To specify that the data is to be encoded using the UNICODE CCSID specified during installation, enter:

```
CCSID UNICODE
```

Specifying ROSHARE

By specifying ROSHARE you can indicate to DB2 how the database will be shared using read-only data. Valid arguments for the ROSHARE keyword are OWNER and READ.

To specify that the database will be shared with only the current server allowed to update it, enter:

```
ROSHARE OWNER
```

To indicate that the current server is to have read-only access to the database through shared read-only data, enter:

```
ROSHARE READ
```

Specifying AS-WORKFILE or AS-TEMP

To specify that this database is a workfile, enter:

```
AS-WORKFILE
```

Or

To specify that this database is used for temporary tables only, enter:

```
AS-TEMP
```

To specify the name of the subsystem for which this database is created, enter:

```
AS-WORKFILE FOR membername
```

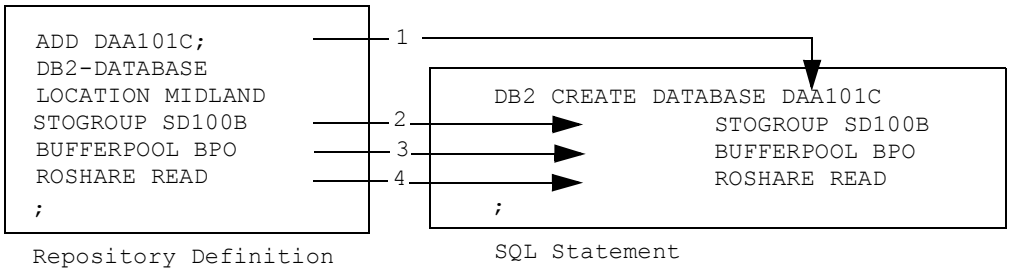
Or

```
AS-TEMP  FOR membername
```

where *membername* is the name of a member.

Example

The DB2-DATABASE member DAA101C is defined in the repository using an ADD command. An SQL statement is generated from the member definition using a DB2 CREATE command.



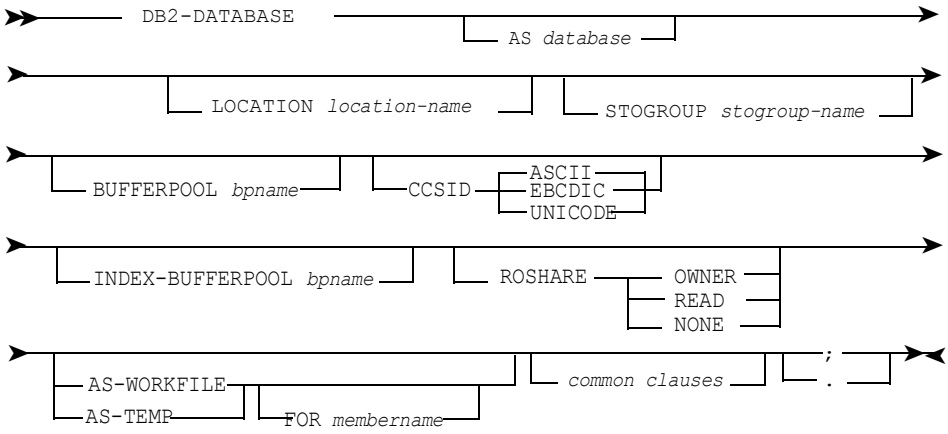
1 The database name DAA1OIC is taken from the member name.

2 The storage group name SD100B is taken from the member definition.

3 The buffer pool name is taken directly from the member definition.

4 The ROSHARE value is taken directly from the member definition.

DB2-DATABASE Syntax



where:

database is the name of a DB2-DATABASE member

location-name is the name of a DB2-LOCATION member

stogroup-name is the name of a DB2-STOGROUP member

membername is string

bpname is one of: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BP10, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP2O, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-DMS

DB2-DMS allows users to document data manipulation statements. Each DB2-DMS member contains one data manipulation statement which may be a delete statement, an insert statement, a select statement or an update statement. To define a DB2-DMS member, enter:

```
DB2-DMS
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-DMS members are optional. The syntax of the DB2-DMS member definition follows.

See "DB2-DMS Syntax" on page 358 for the DB2-DMS member definition syntax.

Documenting an SQL SELECT Statement in DB2-DMS Member

An SQL SELECT can be documented in a DB2-DMS member. This SELECT statement can be a stand-alone member, or can be used with another DB2-DMS or DB2-VIEW member for building complex SELECT statements. For example, to document these SQL SELECT statement:

```
SELECT ALL JAN.SALES, EMP.NAME
  FROM JAN, EMPTABLE AS EMP
  WHERE JAN.EMPNO = EMP.EMPNO
  GROUP BY EMP.NAME
UNION ALL
SELECT ALL  FEB.SALES, EMP.NAME
  FROM FEB, EMPTABLE AS EMP
  WHERE FEB.EMPNO = EMP.EMPNO
  GROUP BY EMP.NAME ;
```

Enter:

```
DB2-DMS
  SELECT ALL
    CONTAINS SALES,
             NAME TABLE EMP GROUP-BY
    FROM JAN, EMPTABLE CORRELATION-NAME EMP
    WHERE 'JAN.EMPNO = EMP.EMPNO'
  UNION ALL
  SELECT ALL
    CONTAINS SALES,
             NAME TABLE EMP GROUP-BY
    FROM FEB, EMPTABLE CORRELATION-NAME EMP
    WHERE 'FEB.EMPNO = EMP.EMPNO'
```

In more complex SQL SELECT statements, the FROM clause may reference a result table from a SQL SELECT statement. Such embedded SELECT statements must be held in another DB2-DMS member to which the first dms member refers:

```
SELECT ALL JAN.SALES, EMP.NAME
  FROM TABLE(SELECT SALES, EMPNO
             FROM FIRSTQTR
             WHERE DATE BETWEEN 0012002 AND 312002) AS JAN,
             EMPTABLE AS EMP
  WHERE JAN.EMPNO = EMP.EMPNO
```

To define this SELECT statement, enter:

```
DB2-DMS
  SELECT ALL
  CONTAINS SALES TABLE JAN, NAME TABLE EMP
  FROM dms-name-2 CORRELATION-NAME JAN,
       EMPTABLE CORRELATION-NAME EMP
  WHERE 'JAN.EMPNO = EMP.EMPNO'
```

where *dms-name-2* is defined as:

```
DB2-DMS
  SELECT ALL
  CONTAINS SALES, EMPNO
  FROM FIRSTQTR
  WHERE 'DATE BETWEEN 0012002 AND 312002'
```

Note:

The general rule is, if parentheses enclose the SELECT statement, the embedded SELECT must be defined in another DB2-DMS member.

A complex SELECT statement with UNIONs can have parentheses used to change the order of execution of the SELECT statements. DB2 will execute the SELECT that is within parentheses first before executing the remainder. This has the same result as if the embedded SELECT had been coded first in the series of SELECT statements. For example, to define these SQL SELECT statement:

```
SELECT ALL MAR.SALES, EMP.NAME
  FROM MAR, EMPTABLE AS EMP
  WHERE MAR.EMPNO = EMP.EMPNO
UNION ALL
  (SELECT ALL JAN.SALES, EMP.NAME
    FROM JAN, EMPTABLE AS EMP
    WHERE JAN.EMPNO = EMP.EMPNO
   UNION ALL
   SELECT ALL  FEB.SALES, EMP.NAME
    FROM FEB, EMPTABLE AS EMP
    WHERE FEB.EMPNO = EMP.EMPNO) ;
```

Enter:

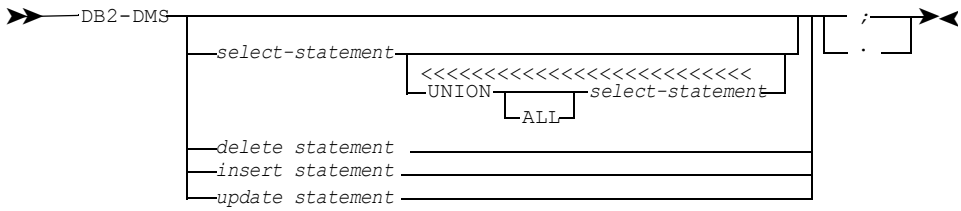
```
DB2-DMS
  SELECT ALL
    CONTAINS SALES, NAME TABLE EMP
    FROM MAR, EMPTABLE CORRELATION-NAME EMP
    WHERE 'MAR.EMPNO = EMP.EMPNO'
  UNION ALL
  SELECT DMS dms-name-4
```

where *dms-name-4* is a DB2-DMS member defined as:

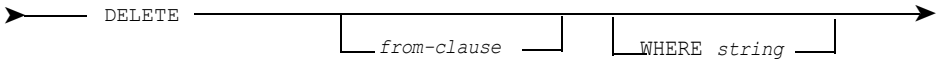
```
DB2-DMS
  SELECT ALL
    CONTAINS SALES, NAME TABLE EMP
    FROM JAN, EMPTABLE CORRELATION-NAME EMP
    WHERE 'JAN.EMPNO = EMP.EMPNO'
  UNION ALL
  SELECT ALL
    CONTAINS SALES, NAME TABLE EMP
    FROM FEB, EMPTABLE CORRELATION-NAME EMP
    WHERE 'FEB.EMPNO = EMP.EMPNO'
```

Since each dms member can also point to successive dms members, it is possible to store very complex SELECT statements with numerous embedded SELECT statements and many levels of parentheses.

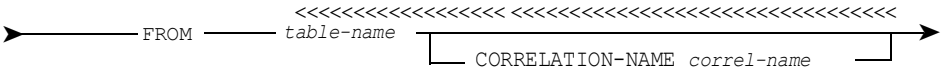
DB2-DMS Syntax



where *delete statement* is:



where *from-clause* is:



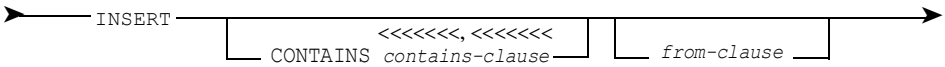
where:

table-name is the name of a DB2-TABLE member.

correl-name is an identifier, of no more than 18 characters.

string is the string of text to be deleted.

insert statement is:



where:

from-clause is as defined above

contains-clause is:

```
➤──┬──────────────┬─ single-column-clause ──┬─ column-attrs ──➤
   ├─ (integer) ──┘                         │
   └─ group-name EXPAND ────────────────────┘
```

where *single-column clause* is:

```
➤──┬─ item-name ─┬─ version ─┬──┬─ KNOWN-AS local-name ─┬──➤
   │             │           │  │                       │
   └─ group-name ────────────┘  └───────────────────────┘
```

where:

item-name is the name of an ITEM member

version is an integer in the range 1 to 15

local-name is the name of the column

group-name is the name of a GROUP member.

column-attrs is:

```
➤──┬──────────────────────
```

where:

dms is the name of a DB2-DMS member.

contains-clause is defined below.

table is the name of a DB2-TABLE member.

view is the name of a DB2-VIEW member.

dms is the name of a DB2-DMS member.

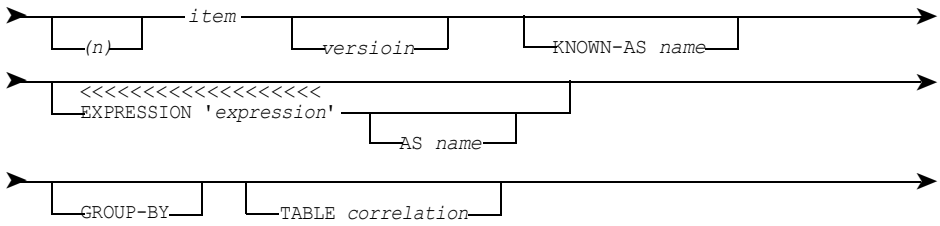
type is one of these JOIN types: INNER, RIGHT OUTER, LEFT OUTER, or FULL OUTER.

cond is the JOIN condition, of up to 255 characters.

name is the correlation name of the table, of up to 18 characters.

string is a string of up to 255 characters.

where *contains-clause* is:



where:

EXPRESSION '*expression*' is an expression of no more than 255 characters, and is used when the column is defined by an expression. However, an item name is still required in order to establish the datatype of the result from the expression. When multiple lines of expressions are entered, each expression must be preceded by the keyword EXPRESSION.

name is the optional name to be given this column.

where:

single-column-clause is as defined above.

column-attrs is:

```
➤──┬──────────────────────────┬──┬──────────────┬──────────────────────────➤
   │ EXPRESSION string        │  │ GROUP-BY     │
   └──────────────────────────┘  └──────────────┘
➤──┬──────────────────────────────────────────────────────────────────────➤
   │ TABLE correl-name                                                    │
   └──────────────────────────────────────────────────────────────────────┘
```

where:

string is a string of 1 to 255 characters, delimited.

correl-name is as defined above.

(*integer*) is a number from 1 to 300.

group-name is as defined above.

where:

single-column clause is as defined above.

(*integer*) is a number from 1 to 300.

group-name is as defined above.

integer is a number from 1 to 18.

where:

item-name is as defined above.

group-name is as defined above.

local-name is as defined above.

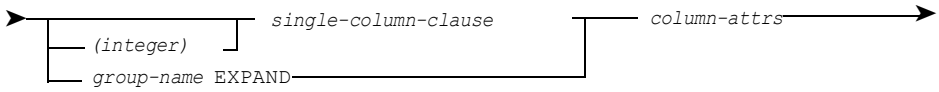
from-clause is as defined above.

string is a string of 1 to 255 characters, delimited.

update-statement is:

```
➤──────────────────────────────────────────────────────────────────────────➤
   UPDATE                                                                 │
                                                  <<<<<<<<<,<<<<<<<<<      │
                                                  CONTAINS contains-clause │
➤──┬──────────────────────────┬──┬──────────────────────────────────────➤
   │ order-by-clause          │  │ from-clause                          │
   │ update-clause            │  │                                      │
➤──┬──────────────────────────┬──┬──────────────────────────────────────➤
   │ WHERE string             │  │ HAVING string                        │
```

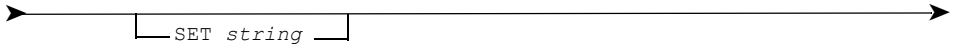
where *contains-clause* is:



where:

single-column-clause is as defined above.

column-attrs is:



where:

string is a string of 1 to 255 characters, delimited.

(*integer*) is a number from 1 to 300.

group-name is as defined above.

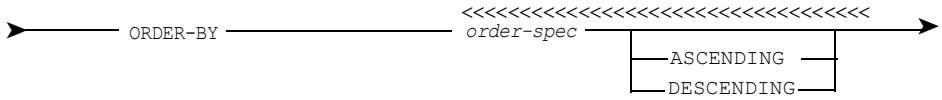
order-by-clause is as defined above.

update-clause is as defined below.

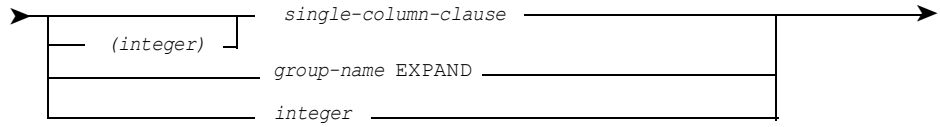
from-clause is as defined above.

string is a string of 1 to 255 characters, delimited.

order-by-clause is:

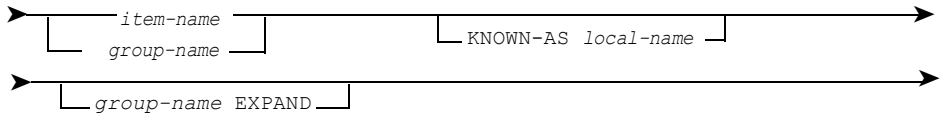


order-spec is:



update-clause is:

where *update-specification* is:



Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of `common-clauses`.

DB2-FUNCTION

DB2-FUNCTION defines user-defined DB2 functions. All 4 different types of functions can be defined:

External Scalar. The function is an external executable that is registered with a database server and returns a scalar value. An external function is defined by specifying an external name for the function. A scalar function is defined by specifying the clause "RETURNS TYPE item" where item is the name of a repository ITEM that defines the scalar value returned by the function.

External Table. The function is an external executable that is registered with a database server and returns a complete table. An external function is defined by specifying an external name for the function. A table function is defined by specifying the clause "RETURNS TABLE `table`" or a "RETURNS COLUMNS `columns`" where `table` is the name of a repository DB2-TABLE member that defines the returned table, and `columns` is a list of column names that comprise the columns of the table returned by the function.

Sourced. The function is implemented by invoking another function (either built-in, external, or sourced) that is already registered with a database server. This type of function is specified by the presence of the source clause in its definition.

SQL Scalar. The function content is specified in the RETURN clause of the CREATE FUNCTION statement. This type of function is new with DB2 Version 7.0. This function is a user-defined SQL function and is identified by LANGUAGE SQL, RETURNS TYPE item (scalar), and the RETURN 'string' clauses in its definition. The RETURN clause is new and unique to this type of function, and describes what the function does, which is different from the RETURNS clause which describes the datatype of the value returned by the function. The RETURN clause is required for SQL scalar functions when the export command DB2 CREATE is used to generate the SQL statement CREATE FUNCTION.

See "DB2-FUNCTION Syntax" on page 367 for the DB2-FUNCTION member definition syntax.

Defining a Function

To define a DB2 function, enter:

```
DB2-FUNCTION
```

Followed by the attributes for this function.

Defining an Owner

To define the owner of this function, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the function. This clause is required for the successful generation of SQL statements, but can be overridden by an SQLID clause in the DB2 export command.

Defining a Comment

To define a comment for this function, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is the comment for this function. This value is used to generate the SQL COMMENT ON statement when executing an export command.

Defining Parameters

To define parameters consisting of a list of items that are passed to this function, enter:

```
PARAMETERS parm-1, parm-2,…
```

where *parm-1* and *parm-2* are names of ITEM, DB2-TABLE, or DB2-VIEW members which describe the parameters being passed.

You can enter these additional attributes:

- CCSID value
- Whether the item is a pointer to a LOB type item (AS LOCATOR)

Defining the LUNAME and LOADMOD name

For DB2 Version 5.0 and earlier, the LUNAME was acquired when function and procedure definitions were imported from DB2. The DB2 catalog table SYSIBM.SYSPROCEDURES which contains this attribute is not used in later releases of DB2. For this reason, LUNAME may not apply for your installation.

To define an LUNAME of the system that issued the SQL CALL statement to invoke this function, enter:

```
LUNAME name
```

where *name* is the luname of the system.

To define a LOADMOD name of the MVS load module that DB2 should load to satisfy the request for the stored procedure, enter:

```
LOADMOD name
```

where *name* is the 8 character member name of the MVS load module.

Defining the LINKAGE Attribute

To define the linkage convention used to pass parameters to the function, enter:

```
LINKAGE NULL
```

if the SIMPLE WITH NULLS convention is used where an indicator array is passed to the stored procedure. Null input parameters are allowed.

Enter:

```
LINKAGE BLANK
```

where the SIMPLE linkage convention is used when input parameters cannot be null.

Defining Results Returned by the Function

To define the data or results returned by the function, determine the type of data returned as one of these:

- A scalar value defined as an item, enter:

  ```
  RETURNS TYPE item
  ```

  where *item* is the name of an ITEM member that describes the returned data.

- A table defined as a DB2-TABLE member, enter:

  ```
  RETURNS TABE table
  ```

  where *table* is the name of a DB2-TABLE member that describes the table returned.

- A table defined as a list of column (item) members, enter:

  ```
  RETURNS COLUMNS col1, col2
  ```

  where *col1* and *col2* are names of ITEM members that comprise the returned table columns.

Defining Attributes

All the rules for consistent attribute values that apply in DB2 also apply to the definition of DB2-FUNCTION members, but they are only enforced when an export command such as DB2 CREATE is issued for the member. For example, when LANGUAGE JAVA is specified, DB2 requires that the PARAMETER-STYLE must also be JAVA. A function may be defined with LANGUAGE JAVA that does not specify PARAMETER-STYLE JAVA, but this error is flagged when the function is used to generate an SQL statement to export this definition to DB2.

Here is the current list of attribute dependencies:

- When LANGUAGE is JAVA, the following apply:
  - PARAMETER-STYLE must be JAVA
  - EXTERNAL-NAME must be specified
  - FINAL-CALL NO must be specified
  - DBINFO NO must be specified
  - SCRATCHPAD must not be specified
  - RUNOPTS must not be specified
- Do not specify EXECUTES UPDATE-SQL when ALLOW-PARALLEL is specified.

There are many similarities between the DB2-FUNCTION and DB2-PROCEDURE member types. The main difference is the presence of the RETURNS clause in the DB2-FUNCTION definition. For this reason, if a DB2-FUNCTION is defined without a RETURNS clause, it is treated as a DB2-PROCEDURE member type when an export command is issued for it.

DB2-FUNCTION Syntax

```
>>-DB2-FUNCTION-+-------------+-+-----------------+->
                |-AS member---| |-CREATOR-OWNER user-|
>-+-----------------------+-------------------------->
  |-DB2-COMMENT 'comment'-|
>-+-PARAMETERS-+-<<<<<<<<<<<<<<<<,<<<<<<<<<<<<<<<-+-+-+-------------+-+-----------+-+-----------+-+->
               | |-item-name-+-+-CCSID-+-ASCII---+-+-+-EBCDIC---+-+-+-AS LOCATOR-+-+
               | | |         | |       |-EBCDIC--| | |-UNICODE-| | |           | |
               | | |         | |       |-UNICODE-| | |           | |           | |
               | | |-table-name-------------------| |           | |           | |
               | | |-view-name--------------------| |           | |           | |
>-RETURNS-+-TYPE-item-+-+-------------+-+-CCSID-+-ASCII---+-+-+-AS LOCATOR-+-+-+->
          | |         | |-CAST-FROM cast-| | |-EBCDIC--| | |           | | |
          | |         | |                | | |-UNICODE-| | |           | | |
          | |-TABLE table-+-+-----------+-+-+-----------+-+-+-----------+-+-+
          | |             | |-AS LOCATOR-| | |           | |           | | |
          | |             | |            | | |           | |           | | |
          | |-COLUMNS column-<<<<<<<<<< , <<<<<<<<<<-+-+-+-----------+-+-+-+
          |                                           | |-AS LOCATOR-| | |
                                                      | |            | | |
```

where:

comment is a comment of no more than 254 characters. This comment is used to generate the SQL COMMENT ON command for this function.

table-name is the name of a DB2-TABLE that defines the result table that is returned by this function.

view-name is the name of a DB2-VIEW that defines the result table that is returned by this function.

member is the name of a repository member.

user is the name of a DB2-USER member.

The CCSID for parameters must be defined individually for each parameter.

options:

```
>-+-+-----------+-+-+-----------+-+-+-----------+-+-+-----------+-+-+-----------+-+-+-----------+-+->
  | |-LANGUAGE-+-ASM-----+-+-+-PARAMETER-STYLE-+-DB2SQL---+-+-+-SPECIFIC-NAME name-+-+-+-EXTERNAL-NAME name-+-+-+-DETERMINISTIC-+-YES-+-+-+
  | |          | |-COBOL--| | |                | |-JAVA----| | | |                | | | |                | | | |           | |-NO--| | | |
  | |          | |-C------| | |                | |         | | | |                | | | |                | | | |           | |     | | | |
  | |          | |-PL1----| | |                | |         | | | |                | | | |                | | | |           | |     | | | |
  | |          | |-JAVA---| | |                | |         | | | |                | | | |                | | | |           | |     | | | |
  | |          | |-SQL----| | |                | |         | | | |                | | | |                | | | |           | |     | | | |
```

```
VARIANT [YES|NO]
EXTERNAL-SECURITY [YES|NO]
CARDINALITY integer
CALLED-ON-NULL-INPUT [YES|NO]
EXECUTES [NO SQL|READ-SQL|UPDATE-SQL|SQL]
EXTERNAL-ACTION [YES|NO]
SCRATCHPAD integer
FINAL-CALL [YES|NO]
CCSID [ASCII|EBCDIC|UNICODE]
FENCED [YES|NO]
ALLOW-PARALLEL [YES|NO]
DBINFO [YES|NO]
COLLID 'id'
WLM-ENVIRONMENT [NONE|SAME|NAME name|'string']
ASUTIME integer
STAY-RESIDENT [YES|NO]
PROGRAM-TYPE [MAIN|SUB]
SECURITY [DB2|USER|DEFINER]
RUNOPTS 'string'
STATIC-DISPATCH
SPECIAL-REGISTERS [DEFAULT|INHERIT]
source-clause
RETURN 'expression'
```

source-clause:

```
SOURCE [SPECIFIC name|FUNCTION name [<<<<<,<<<<< PASSING member-name]]
```

where:

name is the specific or function name of the function.

name2 is the external name of the function.

integer is the estimated number of rows the function returns.

integer2 is the length of scratchpad area allocated for the function, from 0 to 32,767.

id is the name of a package collection ID to be used when the function is executed.

name3 is the MVS workload manager environment name.

string is the MVS workload manager environment name.

integer3 is the ASUTIME limit, from 0 to 2 GB.

string2 is the language environment runtime options, up to 254 characters.

expression describes the action that is performed by this function, and must be no more than 254 characters in length. This attribute is required only for the SQL scalar type functions.

member-name is the name of an ITEM member that is passed by the function.

These are the defaults:

- PARAMETER-STYLE DB2SQL
- DETERMINISTIC NO
- FENCED
- CALLED-ON-NULL-INPUT NO
- EXECUTES READ-SQL
- EXTERNAL-ACTION YES
- No SCRATCHPAD
- FINAL-CALL NO
- DBINFO NO
- COLLID NO
- No ASUTIME LIMIT
- STAY-RESIDENT NO
- PROGRAM-TYPE SUB
- SECURITY DB2
- INHERIT-SPECIAL-REGISTERS
- If DETERMINISTIC NO, EXTERNAL-ACTION YES, SCRATCHPAD, or FINAL-CALL YES is specified, DISALLOW PARALLEL is the default. Otherwise ALLOW-PARALLEL is the default.

When LANGUAGE is JAVA, the following apply:

- PARAMETER-STYLE must be JAVA
- EXTERNAL-NAME must be specified
- FINAL-CALL NO must be specified
- DBINFO NO must be specified

Do not specify EXECUTES UPDATE-SQL when ALLOW-PARALLEL is specified.

The following are synonyms:

- PARAMETER-STYLE STANDARD-CALL is DB2SQL
- PARAMETER-STYLE SIMPLE-CALL is GENERAL
- VARIANT is NOT DETERMINISTIC
- VARIANT NOT is DETERMINISTIC
- NULL-CALL is CALLED-ON-NULL-INPUT

DB2-INDEX

DB2-INDEX defines DB2 indexes in the repository.

Refer to "DB2-INDEX Syntax" on page 387 for the syntax of DB2-INDEX member definition.

To define a DB2 INDEX, enter:

```
DB2-INDEX
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-INDEX members are optional. However for the successful generation of SQL statements you must define the following:

- For ALTER INDEX statements define the CREATOR-OWNER clause
- For CREATE INDEX statements define the CREATOR-OWNER and
- ON clauses, and if the PARTITION clause is defined, the CLUSTER keyword
- For DROP INDEX statements define the CREATOR-OWNER clause

Each column is usually represented by an ITEM member, but can be represented by a GROUP member. GROUP members are useful because they can contain several ITEMs.

To specify the ITEM and GROUP members that represent the indexed columns, use the CONTAINS clause. You can define columns:

- Individually, so that one ITEM or GROUP member defines one column
- In sets, so that the same ITEM or GROUP member defines several columns, with identical attributes.
- In cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

DB2-INDEX repository definitions can be generated automatically if you use the Workbench Design Area (WBDA) facilities for DB2 database design.

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of generating DB2-INDEX definitions from the WBDA.

Refer to Chapter 4, "Repository Definition," on page 91 for details of the derivation of DB2 index names.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-INDEX or SQL-INDEX member definitions using an AS clause.

Defining an Owner

To define the owner of an index, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the index.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

This clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Note:

The DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for indexes.

Defining a Clustered Index

To define a clustered index, enter:

```
CLUSTER
```

Partitioned indexes must also be clustered.

If you do not include the CLUSTER keyword in a DB2-INDEX that contains a PARTITION clause, it is automatically generated in SQL CREATE INDEX statements. A warning message is also generated, to remind you to include the CLUSTER keyword in the member definition.

Specifying Index Type

To specify the type of index, enter:

```
TYPE n
```

where *n* is 1 or 2.

Note:

An error message is issued during SQL generation if the type specified in the member is type 1 and the index is defined as UNIQUE WHERE-NOT-NULL.

Defining a Unique Index

To define that indexed columns in the table being indexed do not have duplicate entries, enter:

```
UNIQUE
```

In DB2, when a single column in a table is being indexed, then each value can appear once only in the indexed column. When more than one column in a table is being indexed, then any given set of values can appear once only in the indexed columns.

In a column of the key that can contain null values, to specify that two or more null values are regarded as unequal, enter:

```
WHERE-NOT-NULL
```

WHERE-NOT-NULL can only be specified on a type-2 index.

Defining the Table to be Indexed

To define the table to which the index refers, enter:

```
ON table
```

where *table* is the name of a DB2-TABLE or SQL-TABLE member.

On encoding, the member specified in the ON clause is checked to ensure that it is a DB2-TABLE or SQL-TABLE member.

The ON clause must be present for the successful generation of SQL CREATE INDEX statements.

Specifying the Form Description that Defines the Data Type of the Columns to be Indexed

When you index a table, every column name generated by the DB2-INDEX member must have a corresponding column name generated by the DB2-TABLE member that is being indexed.

To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns to be indexed in the table, enter one of the following form keywords:

```
ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

For example, a DB2-INDEX member containing the following lines:

```
ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to these 2 ITEM members:

```
ITEM 1
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

```
ITEM2
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-INDEX definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns. Therefore the column generated from ITEM 1 has a data type of CHAR and the column generated from ITEM has a data type of DECIMAL.

If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the Manager Products defaults apply. For further details refer to *ASG-Manager Products Source Language Generation*.

Refer to Appendix C, "Defining and Generating DB2 Member Types," on page 519 for further details of documenting the columns of tables and generating data types.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of form keywords and form description in ITEM and GROUP member definitions.

Specifying the ITEMs or GROUPs that Define Columns

To specify the ITEM or GROUP members that define columns, enter:

```
CONTAINS member-list
```

where *member-list* is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either ITEMs or GROUPs. Duplicate column names are not permitted by DB2, therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

where:

item is the name of an ITEM member.

version is an integer in the range 1 to 15, and defines the form description version that you want to use.

For example:

```
HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description in the ITEM member STOCK- LIST is used a the column data type.

When you use the SIZE and RECALCULATE commands, the data type of columns is used to calculate the size of an index.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

where:

integer is the number of columns to be derived from the member, within brackets

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by 1, the second by 2, and so on.

For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST 1, STOCK_LIST 2, STOCK_LIST_3, and STOCK_LIST_4. The attributes, such as data type, are the same for each of the four columns.

When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where a DB2 command applied to the DB2-TABLE specifies the EXPAND keyword, then each ITEM within a GROUP generates a separate column.

Naming Columns

You can explicitly name a column if you do not want its name to be generated from the ITEM or GROUP name or alias.

To define the name of an indexed column in a table, enter:

```
KNOWN-AS local-name
```

where *local-name* is a string of no more than 18 characters.

For example:

```
CONTAINS IT-INCOMING KNOWN-AS STOCK_IN
```

defines that the ITEM member IT-INCOMING generates an indexed column called STOCK_IN.

If you use the KNOWN-AS clause to name a set of columns, the local name is duplicated for each column. To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by _1, the second by _2, and so on.

For example:

```
CONTAINS (3) IT-Q1 KNOWN-AS MONTH
```

generates three columns from the ITEM member IT-Q1, named MONTH_1, MONTH_2, and MONTH_3.

Refer to Chapter 4, "Repository Definition," on page 91 for further details of the generation of column names.

Specifying that Each ITEM Contained in a GROUP Defines One Column

If you want each of the ITEMs contained in a GROUP to represent a column, enter:

```
CONTAINS group EXPAND
```

where *group* is the name of a GROUP member.

For example entering: `CONTAINS AREA-DEPOT EXPAND` generates four columns when the GROUP, AREA-DEPOT, contains four ITEMs.

However, if the NO-EXPAND keyword is specified in a DB2 command the EXPAND keyword in the member definition is overridden and the GROUP generates a single column.

Note:

You cannot define a KNOWN-AS clause with expanded GROUPs. If you represent a column with a GROUP member, but name the column using a KNOWN-AS clause, duplicated column names are generated if you use DB2 commands that include the EXPAND keyword.

If the GROUP is nested, that is it contains other GROUPs, each of these is also expanded so that all ITEMs are used to define columns. Nesting can continue to any depth and is only limited by the amount of memory available.

Where a set of columns is derived from an expanded GROUP, each contained ITEM or GROUP is repeated the number of times specified. If a GROUP contains an ITEM with its own repeating factor, the ITEM is also repeated the number of times specified.

For example, a DB2-INDEX defined as:

```
CONTAINS (2) AREA-DEPOT EXPAND
```

where the GROUP, AREA-DEPOT, contains:

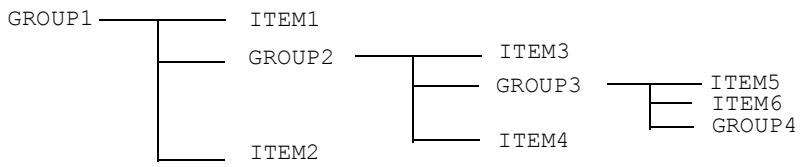
```
(2)      WAREHSE-A
         WAREHSE-B
(3)      LOCALSTK-A
```

generates 4 columns from WAREHSE-A, 2 columns from WAREHSE-B, and 6 columns from LOCALSTK-A, and names them as follows:

```
WAREHSE_A_1
WAREHSE_A_2
WAREHSE_B_1
LOCALSTK_A_1
LOCALSTK_A_2
LOCALSTK_A_3

WAREHSE_A_3
WAREHSE_A_4
WAREHSE_B_2
LOCALSTK_A_4
LOCALSTK_A_5
LOCALSTK_A_6
```

Expanded GROUPs

The member GROUP 1 contains nested GROUPs, shown in this diagram:



When you generate SQL statements intermediate levels in the data structure, that is GROUP2 and GROUP3, are removed in order to generate the following flat, two-level structure:

```
02 ITEM1
02 ITEM3
02 ITEM5
02 ITEM6
02 GROUP4
02 ITEM4
02 ITEM2
```

Note:

When you apply SIZE or RECALCULATE, GROUP4, is treated as an elementary field as it has no lower level. Its data type defaults to CHAR (1). Intermediate levels, in the above example GROUP2 and GROUP3, can be shown as comments.

Defining Sort Order

To define that the entries in an index are sorted in ascending key order, enter:

```
ASCENDING
```

To define that the entries in an index are sorted in descending key order, enter:

```
DESCENDING
```

If you do not specify a sort sequence, the DB2 default applies.

Defining Storage Space

To define the physical space occupied by an index or partition you can either:

- Define the VSAM catalog it is to use
- Define the storage group it belongs to

To define the VSAM catalog the index or partition is to use, enter:

```
VCAT catalog
```

where *catalog* is the name of a VSAM catalog, of no more than 8 characters.

To define the storage group to which the index or partition belongs, enter:

```
STOGROUP stogroup-name
```

where *stogroup-name* is the name of a DB2-STOGROUP member.

On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.

The storage group defined in the DB2-DATABASE definition is the default used by indexes

- That belong to the database
- That do not have a storage group specified in their own member definition

When you define the STOGROUP clause, you can additionally define further details of primary and secondary storage using the PRIQTY, SECQTY sub-clauses. Both sub-clauses are expressed in Kilobytes.

To specify the amount of primary storage space, enter:

```
PRIQTY p
```

where *p* is the number of kilobytes, and must be in the range 3 to 4194304 inclusive.

To specify the amount of secondary storage space, enter:

```
SECQTY s
```

where *s* is the number of kilobytes, and must be in the range 0 to 131068 inclusive.

Primary and secondary storage space is allocated in the storage area defined either:

- In the VOLUMES clause of the DB2-STOGROUP member named in the STOGROUP clause of the DB2-INDEX member
- In the storage group defined in the STOGROUP clause of the DB2- DATABASE

You can also specify whether or not to erase DB2-defined datasets when the index or partition is deleted in a DROP command.

If you want the datasets to be erased, enter:

```
ERASE YES
```

If you do not want the datasets to be erased, enter:

```
ERASE NO
```

If you do not specify PRIQTY, SECQTY, or ERASE sub-clauses, the DB2 defaults apply.

Defining Free Space

You can accommodate future expansion of an index or partition by defining:

- The frequency with which pages are left free
- The percentage of each page that is left free

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE fn
```

where *fn* is an integer in the range 0 to 255.

For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

To define the percentage space kept free on a page, when an index or partition is loaded or reorganized, enter:

```
PCTFREE pn
```

where *pn* is an integer in the range 0 to 99

If you do not define FREEPAGE or PCTFREE clauses the DB2 defaults apply.

Defining Group Buffer Pool Usage

To define that all pages are to be cached in the group buffer pool, enter:

```
GBPCACHE ALL
```

To define that only updated pages are to be cached in the group buffer pool, enter:

```
GBPCACHE CHANGED
```

To define that no pages are to be put into the group buffer pool, enter:

```
GBPCACHE NONE
```

Defining the Ability to Recover with the COPY Utility

Specifying `COPY YES` allows recovery of the index and specifying `COPY NO` suppresses the ability to recover.

Defining Index Partitions

To define that an index is to have a partition, enter:

```
PARTITION
```

You can optionally define a number, details of storage, free space allocation and key values, for each partition.

To generate SQL CREATE INDEX statements successfully from DB2-INDEX members containing a PARTITION clause, you must define a NUMBER and KEY clause for each PARTITION.

To give the partition a number, enter:

```
NUMBER n
```

where *n* is an integer in the range 1 to 64. If you do not define the numbers of partitions, they are automatically generated by the DB2 CREATE command, commencing with 1, and increasing in increments of 1.

To define that the partitions of an index have a key value, enter:

```
KEY 'key-val'
```

where *key-val* is the highest value, within delimiters, that a column in the partition can have.

If you are indexing more than one column, each key value must be enclosed in quotes and separated by a comma. The first key value corresponds to the first index column, the second value to the second index column and so on.

If the key value is numeric it must be enclosed in single quotes. If the key value is a character string, or includes a character, it must be enclosed in single quotes within double quotes. For example:

```
KEY 2525

KEY " 'x2525'"
```

To define the storage space to which the partition belongs, use the VCAT, or STOGROUP, PRIQTY, SECQTY, and ERASE clauses described in "Defining Sort Order" on page 377.

To define how much space is left free in the partition, use the FREEPAGE and PCTFREE clauses described in "Defining Storage Space" on page 377.

If you do not define storage or free space for partItions individually, they automatically take space from that specified for the index as a whole.

Defining the Locking Parameter

Each physical page of an index may be divided into 1, 2, 4, 8, or 16 subpages. Each subpage is a unit of locking.

To define subpages and therefore the locking unit, enter:

```
SUBPAGES division
```

where *division* is one of these integers:

```
1
2
4
8
16
```

If you do not define a locking parameter, none is generated, and the DB2 default applies.

Defining an Associated Buffer Pool

To define the buffer pool that indexes use, enter:

```
BUFFERPOOL bufferpool-name
```

where *bufferpool-name* is one of the following buffer pools: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BPl0, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49.

If you do not define a buffer pool in a DB2-INDEX, the default defined in the DB2-DATABASE to which the index belongs, applies.

Specifying Maximum Piecesize for Non-partitioned Indexes

To specify the piecesize, enter:

```
PIECESIZE ps psunits
```

where:

ps is an integer that must be a power of two, the range of which is dependent on the units defined as *psunits*.

psunits can be either K, M, or G and indicates the units for the value specified in *ps*:

- *K* indicates that the *ps* value is to be multiplied by 1024 to specify the maximum piecesize in bytes. *ps* must be an integer value to the power of two between 256 and 4,194,304.
- *M* indicates that the *ps* value is to be multiplied by 1,048,576 to specify the maximum piecesize in bytes. *ps* must be an integer value to the power of two between 1 and 4096.
- *G* indicates that the *ps* value is to be multiplied by 1,073,741,824 to specify the maximum piecesize in bytes. *ps* must be an integer value to the power of two between 1 and 4.

These are the valid values for piecesize:

256K		
512K		
1024K	(or 1 MB)	
2024K	(or 2 MB)	
4096K	(or 4 MB)	
8192K	(or 8 MB)	
16384K	(or 16 MB)	
32768K	(or 32 MB)	
65536K	(or 64 MB)	
131072K	(or 128 MB)	
262144K	(or 256 MB)	
524288K	(or 521 MB)	
1048576K	(or 1024 MB)	(or 1 GB)
2097152K	(or 2048 MB)	(or 2 GB)
4194304K	(or 4096 MB)	(or 4 GB)

The limit is 64GB for DB2 Version 6.0 or later.

Defining the Datasets are Left Open or Closed After Use

To define that the dataset, on which an index resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

If you do not define a CLOSE clause the DB2 default applies.

Defining a Dataset Password

To define a password for the VSAM dataset, on which an index resides, enter:

```
DSETPASS password
```

where *password* is a VSAM dataset password of no more than 8 characters.

Indicating Deferred Index Construction

To indicate whether the index is built during the execution of the CREATE INDEX statement, enter:

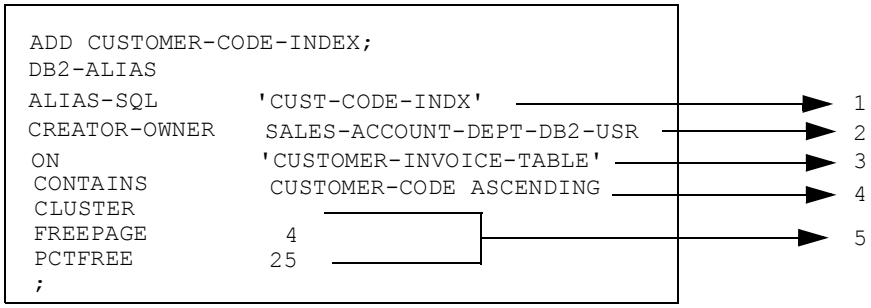
```
DEFER NO
```

indicating that the index is built, or

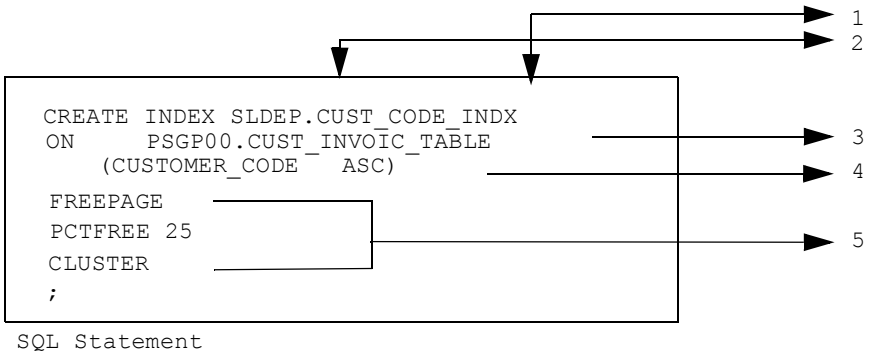
```
DEFER YES
```

indicating that the index is not built. If the table is populated, the index is placed in a recover-pending state to indicate that the index must be recovered by the RECOVER INDEX utility.

Example of an Unpartitioned DB2-INDEX Definition and Generated SQL Statement



Repository definition



SQL Statement

1 The unqualified DB2 name for the index is taken from the SQL ALIAS defined in the member definition statement.

2 The SQL ALIAS of the CREATOR-OWNER (member SALES-ACCOUNT-DEPT-DB2-USR) is used to qualify the index name.

3 The member name of the table to be indexed is CUSTOMER-INVOICE-TABLE, which is generated as CUST_INVOIC_TABLE by name reduction. It is qualified by the table owner's authorization ID, which is derived from the SQL ALIAS of the DB2-USER member, which represents the authorization ID.

4 The CONTAINS clause is converted to a DB2 column-specification by using the SQL ALIAS of member CUSTOMER-CODE and adding the DB2 abbreviation ASC for ASCENDING.

5 The CLUSTER keyword and FREEPAGE and PCTFREE parameters are taken from the member definition.

Example of a Partitioned DB2-INDEX Definition and Generated SQL Statement

In this example the indexed table is in a table space that contains three partitions. Refer to the documentation of the DB2-TBSPACE member type for the example definition of this table space.

The first partition is for all customers whose customer-numbers start with P1, as they are more active than the average. The second partition is for customer 5 whose customer-numbers start in the range P2 to P5. The last partition is for the rest, as these are inactive customers and fewer insertions are expected.

```
ADD CUSTOMER-CODE-INDEX;
DB2-INDEX
ALIAS-SQL              'CUST-CODE-INDX'                  1
CREATOR-OWNER          SALES-ACCOUNT-DEPT-DB2-USR        2
ON                     'CUSTOMER-INVOICE-TABLE'          3
CONTAINS               CUSTOMER-CODE ASCENDING           4
  VCAT VSMCAT3
  FREEPAGE 5
  PCTFREE 10
  CLUSTER
  PARTITION NUMBER 1
    KEY "'P1999999'"
    STOGROUP SG-SD100B
       PRIQTY 200
       SECQTY 20
       ERASE NO                                          5
    FREEPAGE 4
  PARTITION NUMBER 2
    KEY "'P59999999
    VCAT VSMCAT3
  PARTITION NUMBER 3
    KEY "'P9999999
    STOGROUP SG-SD100B
    PCTFREE 20
  SUBPAGE
  BUFFERPOOL BP0
  CLOSE NO
;
```

Repository definition

```
CREATE INDEX SLDEP.CUST_CODE_INDX
ON       PSGP00.CUST_INVOIC_TABLE
         (CUST_CODE ASC)

  USING VCAT VSMCAT3
  FREEPAGE 5
  PCTFREE 10
  CLUSTER
  PART 1 VALUES ('P1999999)
   USING STOGROUP SG-SD100
     PRIQTY 200
       SECQTY 20
       ERASE NO
   FREEPAGE 4
  ,PART 2 VALUES ('P5999999')
    USING VCAT VSMCAT3
  ,PART 3 VALUES ('P9999999')
    USING STOGROUP SG-SD100
    PCTFREE 20
  )
  SUBPAGES  1
  BUFFERPOOL BP0
  CLOSE NO
;
```

SQL Statement

1 The unqualified DB2 name for the index is taken from the SQL ALIAS defined in the member definition statement.

2 The SQL ALIAS of the CREATOR-OWNER (member SALES-ACCOUNT-DEPT-DB2-USR) is used to qualify the index name.

3 The member name of the table to be indexed is CUSTOMER-INVOICE TABLE, which is generated as CUST_INVOIC_TABLE by name reduction. It is qualified by the table owner's authorization ID, which is derived from the SQL ALIAS of the DB2-USER member which represents the authorization ID.

4 The CONTAINS clause is converted to a DB2 column-specification by using the SQL ALIAS of member CUSTOMER-CODE and adding the DB2 abbreviation ASC for ASCENDING.

5 All these attributes are generated directly from the clauses in the member definition.

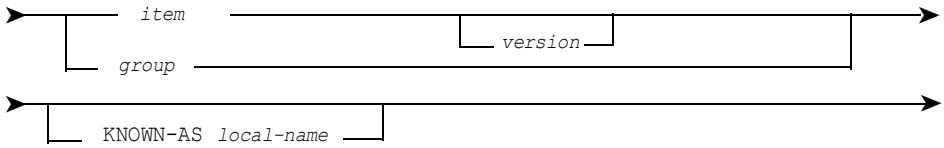
DB2-INDEX Syntax

```
>>-- DB2-INDEX --+-------------+-+-------------------+-+-------------------+-
                 +- AS index -+ +- CREATOR-OWNER user -+

-+-------------------+-+-------+-+-------------+-+-------------------+-
 +- DB2-COMMENT 'comment'-+ +- TYPE -+- 1 -+- 2 -+ +- UNIQUE -+-------------+-+
                                                  +- WHERE-NOT-NULL -+

-+-------------+-+-------------+-+-------------------------------------+-
 +- ON table -+ +- ENTERED-AS -+ +- <<<<<<<<<< , <<<<<<<<<<< -+
                +- HELD-AS -----+ +- CONTAINS contains-clause -+
                +- REPORTED-AS -+
                +- DEFAULTED-AS -+

-+-------------+-+-------------+-+-------------+-+-------------+-+-------+-
 +- using-block -+ +- free-block -+ +- CLUSTER -+ +- DEFER -+- NO -+
                                                            +- YES -+

-+-------------+-+-------+-+-------------------------------------------+-
 +- DEFINE -+- NO -+
            +- YES -+

-+-------------+-+-------------------------------------------------------+-
 +- <<<<<<<<<<<<<<<< <<<<<<<<<<<<<<
```

where:

integer is the number of columns in a set.

single-column-clause is:



where:

item is the name of an ITEM member

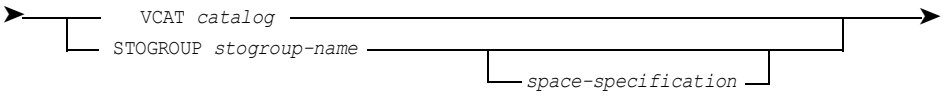
version is an integer in the range 1 to 15

group is the name of a GROUP member

local-name is the name of the column, of no more than 18 characters.

group is as defined above.

using-block is:

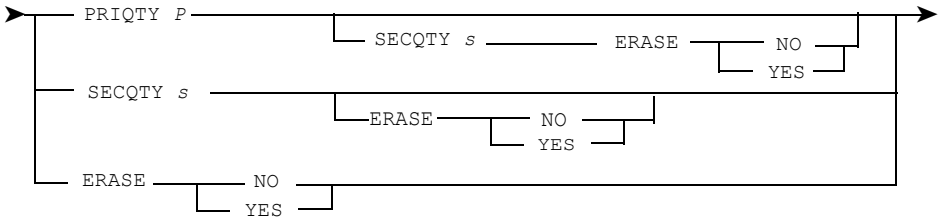


where:

catalog is a VSAM catalog name, of no more than 8 characters

stogroup-name is the name of a DB2-STOGROUP member.

space specification is:

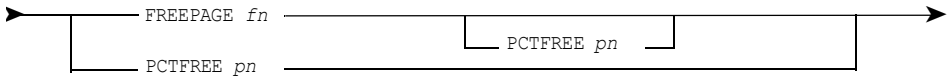


where:

p is an integer in the range 3 to 4194304

s is an integer in the range 0 to 131068.

free-block is:



where:

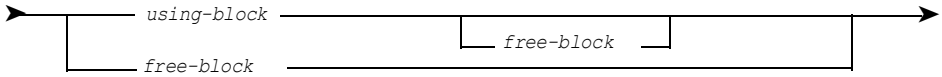
fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99.

n is an integer in the range 1 to 64

key-val is the highest value which the column may contain in the partition

storage is:



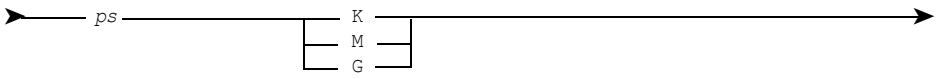
where:

using-block is as defined above

free-block is as defined above.

bpname is one of: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BP10, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K0, BP16K1 BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

maxpsize is:



ps is an integer value that must be a power of two.

If *K* is specified, *ps* must be between 256 and 4194304. The *ps* value will be multiplied by 1024 to specify the maximum piecesize in bytes.

If *M* is specified, *ps* must be between 1 and 4096. The *ps* value will be multiplied by 1048576 to specify the maximum piecesize in bytes.

If *G* is specified, *ps* must be between 1 and 4. The *ps* value will be multiplied by 1073741824 to specify the maximum piecesize in bytes. The limit is 64GB for DB2 Version 6.0 or later.

password is a VSAM dataset password, of no more than 8 characters.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-LOCATION

DB2-LOCATION documents a specific location in a distributed network, which represents a DB2 subsystem.

DB2-LOCATION members are used to generate three-part qualified object names when your DB2 profile is set for three-part name generation. Two-part names are generated by default.

To define a DB2-LOCATION member, enter:

```
DB2-LOCATION
```

The member definition must begin with this member type identifier.

To document the VTAM name of a DB2 location, enter:

```
LUNAME 'luname'
```

where *luname* is a string of no more than eight characters and represents the VTAM logical unit name of the DB2 location. The LUNAME clause is not used to generate SQL statements, but is available to document your DB2 database.

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-LOCATION member definitions using an AS clause.

Refer to Chapter 5, "Export to DB2," on page 105 for details of the DB2 profile.

To generate a location qualifier for table and view names you can:

- Use DB2 commands with the LOCATION keyword (for example DB2 CREATE)
- Use relationships between repository members.

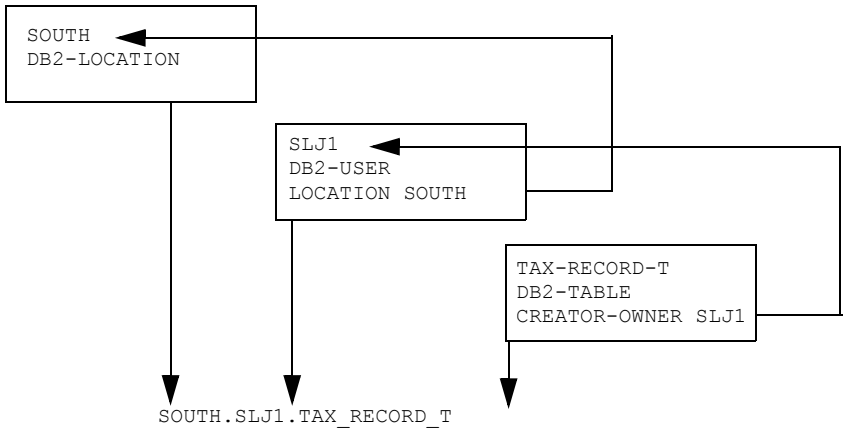
Note:

A location explicitly named in a command overrides one derived from member relationships.

You can generate a location qualifier from the following relationships between members:

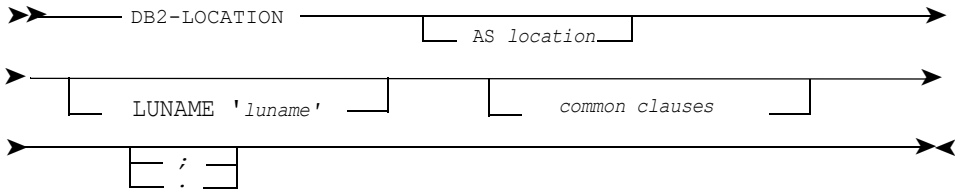
- A DB2-LOCATION member must be named in the LOCATION clause of a DB2-USER member
- The DB2-USER member must be named as the CREATOR-OWNER of the DB2-TABLE or DB2-VIEW being generated.

For example, to generate the three-part table name SOUTH.SLJ1.TAX_RECORD_T, these relationships must exist between members in the repository:



Refer to Chapter 4, "Repository Definition," on page 91 for details of generating location-qualified names. Refer to Appendix A, "Name Reduction Process," on page 511 for details of the name reduction process.

DB2-LOCATION Syntax



where:

`location` is the name of a DB2-LOCATION member.

`luname` is the VTAM name for a location in a network. It is a maximum of 8 characters and must be in delimiters.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of `common clauses`.

DB2-PACKAGE

DB2-PACKAGE defines DB2 application packages in the repository.

Refer to "DB2-PACKAGE Syntax" on page 396 for the syntax of the DB2-PACKAGE member definition.

To define a DB2 package, enter:

```
DB2-PACKAGE
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-PACKAGE members are optional.

A DB2-PACKAGE must be associated with either one SYSTEM, PROGRAM, or MODULE member or with another PACKAGE member. This association is defined in the MEMBER or COPY clauses. Note that on import, unless there is a record on the DB2 CATALOG that a PACKAGE has been produced by a BIND statement with the COPY option, a MEMBER clause is generated by default.

A DB2-PACKAGE must specify the name of a DB2-COLLECTION member to which it belongs. (The COLLECTION clause, like all other clauses, is optional at encode time, but it is essential at generation time.)

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-PACKAGE member definitions using an AS clause.

Defining an Owner

To define the owner of a package, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the plan.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

The clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Defining Bind Parameters

The optional clauses in Table 23 allow you to define parameters within which the BIND or REBIND subcommands produce the application package:

Table 23 Optional Clauses

Clause	Alternatives				
VALIDATE	RUN	BIND			
ISOLATION	RR	RS	CS	UR	NC
RELEASE	COMMIT	DEALLOCATE			
EXPLAIN	YES	NO			
SQLERROR	NOPACKAGE	CONTINUE			
CURRENTDATA	YES	NO			
DEGREE	1	ANY			

You can only define one option for each parameter. The clauses correspond to the keywords in the BIND and REBIND subcommands, defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using he DB2 BIND or DB2 REBIND commands.

Note:

For the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE.

Specifying Dynamic or Static Rules

You can specify whether dynamic or static rules will apply to a dynamic SQL statement at run time, using the DYNAMICRULES keyword. To specify that dynamic rules apply, enter:

```
DYNAMICRULES RUN
```

To specify that static rules apply, enter:

```
DYNAMICRULES BIND
```

For further information on the effects of static rules, refer to the IBM documentation.

Specifying Whether to Keep Dynamic Rules

You can specify whether or not dynamic rules will be kept after the commit point using the KEEPDYNAMIC keyword. To specify that dynamic rules will be kept, enter:

```
KEEPDYNAMIC YES
```

If you do not wish to keep the dynamic rules after commit points, enter:

```
KEEPDYNAMIC NO
```

Specifying Whether to Determine an Access Path

You can specify whether to determine an access path at runtime. To specify that access paths will be determined using default values for input variables, enter:

```
VARS NOREOPT
```

To specify that access paths will be determined using the values of input host variables, enter:

```
VARS REOPT
```

Specifying Whether to Defer Preparation for Dynamic Rules

You can specify whether to defer preparation for dynamic rules that refer to remote objects, or to prepare them immediately. If you do not wish to defer preparation, enter:

```
PREPARE NODEFER
```

If you wish to defer preparation, enter:

```
PREPARE DEFER
```

Specifying the System Environment

You can enable or disable connections to specific environments using the ENABLE and DISABLE keywords. The ENABLE and DISABLE keywords are mutually exclusive. To enable a specific environment, enter:

```
ENABLE ENVIRONMENT environment
```

and to disable a specific environment, enter:

```
DISABLE ENVIRONMENT environment
```

where *environment* is the name of the environment to be enabled or disabled.

If a connection type is disabled, the plan is unable to access that environment type.

You may enable all environments by entering:

```
ENABLE ALL
```

and DB2 accepts this as the default if neither ENABLE nor DISABLE is present in the member definition. You cannot, however, disable all environments.

Whenever you specify an environment type to be enabled or disabled, you may specify an environment name. The name is only allowed if the appropriate environment type has been previously specified.

Specifying a Cache Size

You can specify the size (in bytes) of the authorization cache to be acquired in the EDMPOOL for the plan. To specify a cache size enter:

```
CACHESIZE value
```

where *value* is a number in the range 0 to 4096. The default value is 1024. If you specify a value that is not a multiple of 256, DB2 will round it up to the next highest multiple of 256.

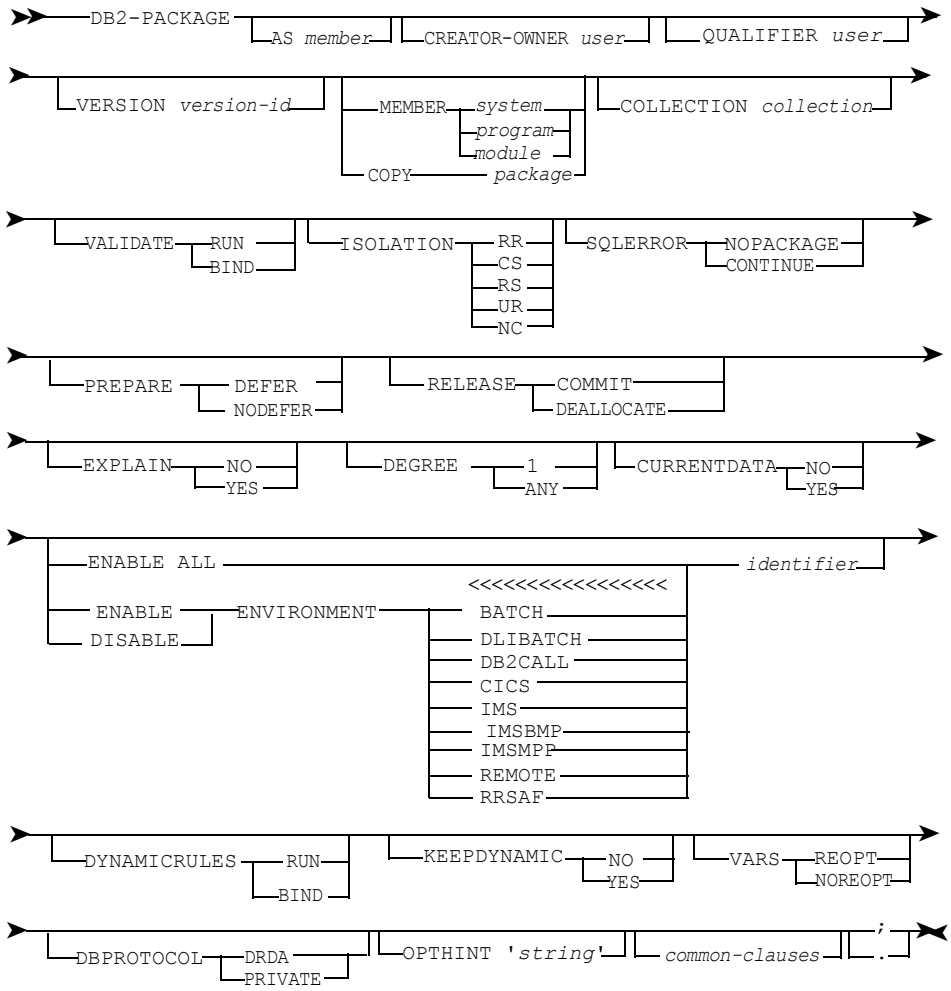
Specifying a Version Identifier

You can specify a version ID for a package so that, when rebinding, an existing version is not overwritten. To specify a version ID enter:

```
VERSION version-id
```

where *version-id* is a delimited string of 1 to 64 characters.

DB2-PACKAGE Syntax



where:

`DBPROTOCOL` specifies which protocol to use when connecting to a remote site that is identified by a three-part name statement. For DRDA, a package must be bound to each remote site that is referenced by the three-part name statements.

`OPTHINT` controls whether query optimization hints are used for static SQL. The '*`string`*' is a hint ID of up to 8 characters used by the optimizer when searching the PLAN_TABLE for rows to use as input to the optimizer.

`package` is the name of a DB2-PACKAGE member.

`user` is the name of a DB2-USER or SQL-USER member.

`version-id` is a string of 1 to 64 characters, delimited.

system is the name of a SYSTEM or MMR-SYSTEM member

program is the name of a PROGRAM member

module is the name of a MODULE member

package is as defined above

collection is the name of a DB2-COLLECTION member.

identifier is:

```
<<<<<<<<<<<<<<<< <<<<<<<<<<<<<<<
```

DB2-PLAN

DB2-PLAN defines DB2 application plans in the repository.

Refer to "DB2-PLAN Syntax" on page 404 for the syntax of the DB2-PLAN member definition.

To define an application plan in the repository, enter:

```
DB2-PLAN
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-PLAN members are optional. However the CREATOR-OWNER and CONTAINS clause must be defined for the successful generation of BIND and REBIND subcommands.

Usually MODULE members are used to represent modules of code containing embedded SQL statements, but PROGRAM, MMR-SYSTEM, or SYSTEM members can also be used. The source code modules are passed through DB2 Precompiler to produce precompiled code modules and Database Request Modules (DBRMs). The BIND or REBIND subcommand is used to bind together the DBRMs and so produce the application plan in the DB2 database. The precompiled code is compiled and link-edited to form the application program that uses the plan to access DB2.

The BIND and REBIND subcommands can be generated from DB2-PLAN definitions using the DB2 BIND and DB2 REBIND commands respectively.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-PLAN member definitions using an AS clause.

Defining an Owner

To define the owner of a plan, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the plan.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

The clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Specifying the Members that Represent DBRMs

To specify the members that represent the database request modules bound into the plan, enter:

```
CONTAINS member-list
```

where *member-list* is the name of one or more MODULE, PROGRAM, MMRSYSTEM, SYSTEM, or DB2-PACKAGE members, separated by commas.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are MODULE, PROGRAM, MMR-SYSTEM, SYSTEM, or DB2-PACKAGE members.

DB2-PACKAGEs can be specified either generically or individually. See "Expanding Packagelist Entries when Importing Plans" on page 302 for further information on generic packagelist.

If generic package names are required, the following packages must be added to the repository:

```
PLAN CONTAINS

PK-ALL-ALL-ALLDB2-PACKAGE
    ALIAS SQL '*'
    COLLECTION CL-*-*

CL-ALL-ALLDB2-COLLECTION
    ALIAS SQL '*'
    LOCATION '*'

LN-ALLDB2-LOCATION
    ALIAS SQL '*'
```

Defining Bind Parameters

The optional clauses in Table 24 allow you to define parameters within which the BIND or REBIND subcommands produce the application plan:

Table 24 Optional Clauses

Clause	Alternatives		
PREPARE	DEFER	NODEFER	
VALIDATE	RUN	BIND	
ISOLATION	RR	CS	UR
ACQUIRE	USE	ALLOCATE	
RELEASE	COMMIT	DEALLOCATE	
EXPLAIN	YES	NO	
CURRENTDATA	YES	NO	
DEGREE	1	ANY	
SQLRULES	DB2	STD	
DISCONNECT	EXPLICIT	AUTOMATIC	CONDITIONAL

You can only define one option for each parameter. The clauses correspond to the keywords in the BIND and REBIND subcommands, defined in the IBM documentation and their meanings are the same. Additional parameters can be defined using the DB2 BIND or DB2 REBIND commands.

Note:

For the successful generation of BIND and REBIND subcommands you must define RELEASE DEALLOCATE with ACQUIRE ALLOCATE.

Specifying Dynamic or Static Rules

You can specify whether dynamic or static rules will apply to a dynamic SQL statement at run time, using the DYNAMICRULES keyword. To specify that dynamic rules apply, enter:

```
DYNAMICRULES RUN
```

To specify that static rules apply, enter:

```
DYNAMICRULES BIND
```

For further information on the effects of static rules, refer to the IBM documentation.

Specifying Whether to Keep Dynamic Rules

You can specify whether or not dynamic rules will be kept after the commit point using the KEEPDYNAMIC keyword. To specify that dynamic rules will be kept, enter:

```
KEEPDYNAMIC YES
```

If you do not wish to keep the dynamic rules after commit points, enter:

```
KEEPDYNAMIC NO
```

Specifying Whether to Determine an Access Path

You can specify whether to determine an access path at runtime. To specify that access paths will be determined using default values for input variables, enter:

```
VARS NOREOPT
```

To specify that access paths will be determined using the values of input host variables, enter:

```
VARS REOPT
```

Specifying a Current Server

To specify a connection to a location for the PLAN, enter:

CURRENTSERVER *location-name*

where *location-name* is the name of a DB2-LOCATION member. During plan allocation, the server's CURRENT SERVER register is set to the location specified. The default is the current DBMS.

Specifying a Cache Size

You can specify the size (in bytes) of the authorization cache to be acquired in the EDMPOOL for the plan. To specify a cache size enter:

CACHESIZE *value*

where *value* is a number in the range 0 to 4096. The default value is 1024. If you specify a value that is not a multiple of 256, DB2 will round it up to the next highest multiple of 256.

Generic Package Lists

DB2 PLAN members can CONTAIN packages with generic names, so that during BIND, the PKLIST generated does not have to refer to each package by name. See "Expanding Packagelist Entries when Importing Plans" on page 302 for more information regarding how to set up these members.

Specifying the System Environment

You can enable or disable connections to specific environments using the ENABLE and DISABLE keywords. The ENABLE and DISABLE keywords are mutually exclusive. To enable a specific environment, enter:

```
ENABLE ENVIRONMENT environment
```

and to disable a specific environment, enter:

```
DISABLE ENVIRONMENT environment
```

where *environment* is the name of the environment to be enabled or disabled.

If a connection type is disabled, the plan is unable to access that environment type.

You may enable all environments by entering:

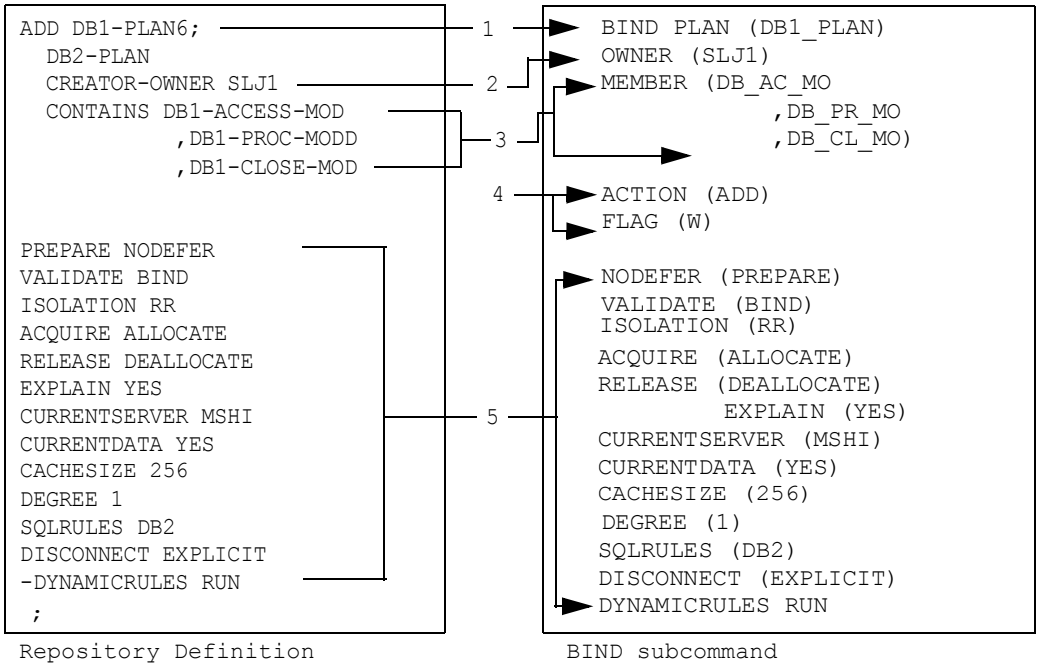
```
ENABLE ALL
```

and DB2 accepts this as the default if neither ENABLE nor DISABLE is present in the member definition. You cannot, however, disable all environments.

Whenever you specify an environment type to be enabled or disabled, you may specify an environment name. The name is only allowed if the appropriate environment type has been previously specified.

Example

The DB2-PLAN member DB1-PLAN6 is defined in the repository using an ADD command. An SQL statement is generated from the member definition using a DB2 BIND command.



1 The plan name is derived from the member name and is reduced to eight characters.

2 The owner name is derived from the CREATOR-OWNER clause.

3 The DBRM names are derived from the CONTAINS clause. Each name is reduced to eight characters.

4 The ACTION and FLAG parameters are derived from the DB2 BIND command.

5 The remaining bind parameters are derived directly from the member definition.

DB2-PLAN Syntax

```
>>──DB2-PLAN──┬──────────────┬──┬──────────────────────┬──┬──────────────┬──>
              └─AS member────┘  └─CREATOR-OWNER user───┘  └─QUALIFIER user─┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────┬──>
  └─CONTAINS──┬──────────────────────┬─┘  └─ISOLATION──┬─RR──┬─┘  └─VALIDATE──┬─RUN──┬─┘
              │ <<<<< , <<<<         │                 │ CS  │                │ BIND │
              ├─module───────────────┤                 └─UR──┘                └──────┘
              ├─program──────────────┤
              ├─system───────────────┤
              └─package──────────────┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──>
  └─PREPARE──┬─DEFER────┬──────────┬─┘  └─ACQUIRE──┬─USE────┬─┘  └─RELEASE──┬─COMMIT───────┬─┘
             └─NODEFER──┘          │                └─ALLOCATE─┘               └─DEALLOCATE───┘
             ├──────────────────────┤
             └─EXPLAIN──┬─YES──┬────┘
                        └─NO───┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──>
  └─CURRENTDATA──NO──┘                └─CACHESIZE value──────┘  └─CURRENTSERVER location─┘

>──┬──────────────────────────────────┬──┬──────────────────────────────────────────────┬──>
  └─CURRENTDATA──NO──┘                └─ENABLE ALL──────────────────────────────────────┘
                                        ├─ENABLE──┬──────────────────────────────────────┐
                                        ├─DISABLE─┘─ENVIRONMENT──┬─<<<<<<<<<<──────────┬─┘
                                        │                        ├─BATCH───────────────┤
                                        │                        ├─DLIBATCH────────────┤
                                        │                        ├─DB2CALL─────────────┤
                                        │                        ├─CICS────────────────┤
                                        │                        ├─IMS─────────────────┤
                                        │                        ├─IMSBMP──────────────┤
                                        │                        ├─IMSMPP──────────────┤
                                        │                        └─RRSAF───────────────┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──>
  └─DEGREE──┬─1────┬──────────────────┬─┘  └─SQLRULES──┬─DB2──┬─┘  └─DISCONNECT──┬─EXPLICIT──────┬─┘
            └─ANY──┘                  │                └─STD──┘                   ├─AUTOMATIC─────┤
                                      ├──────────────────────┤                   └─CONDITIONAL───┘
                                      └─DYNAMICRULES──┬─RUN──┬─┘
                                                      └─BIND─┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──>
  └─DYNAMICRULES──┬─RUN──┬──────────┬─┘  └─KEEPDYNAMIC──┬─NO───┬─┘  └─VARS──┬─REOPT───┬─┘
                  └─BIND─┘          │                   └─YES──┘            └─NOREOPT─┘
                                    ├──────────────────────┤
                                    └─DBPROTOCOL──┬─DRDA────┬─┘
                                                  └─PRIVATE─┘

>──┬──────────────────────────────────┬──┬──────────────────────┬──┬──────────────────────┬──┬──┬──>
  └─DBPROTOCOL──┬─DRDA────┬──────────┬─┘  └─OPTHINT 'string'──┘  └─common-clauses──────┘  └─;──┘  │
                └─PRIVATE─┘          │                                                     └─.──┘
                                     ├──────────────────────┤
                                     └─OPTHINT 'string'─────┘
```

where:

`DBPROTOCOL` specifies which protocol to use when connecting to a remote site that is identified by a three-part name statement. For DRDA, a package must be bound to each remote site that is referenced by the three-part name statements.

`OPTHINT` controls whether query optimization hints are used for static SQL. The '*`string`*' is a hint ID of up to 8 characters used by the optimizer when searching the PLAN_TABLE for rows to use as input to the optimizer.

`member` is the name of a DB2-PLAN member.

`user` is the name of a DB2-USER or SQL-USER member.

`module` is the name of a MODULE member.

`program` is the name of a PROGRAM member.

`system` is the name of a SYSTEM or MMR-SYSTEM member.

package is the name of a DB2-PACKAGE member.

value is a number from 0 to 4096.

location is the name of a DB2-LOCATION member.

identifier is:

```
<<<<<<<<<<<<<<<< <<<<<<<<<<<<<<<
```

You can define privileges on:

- Databases
- Tables
- Plans
- Buffer pools
- Storage groups
- Table spaces
- The DB2 system
- Collections

You can generate SQL GRANT or REVOKE statements from DB2-PRIVILEGE members using the DB2 GRANT and DB2 REVOKE commands.

All privileges, except SYSTEM and USE privileges, give access to particular DB2 objects. For example, a DATABASE privilege gives a user access to a specific, named database, defined in the ON clause.

You can record the grantors and recipients of privileges, and optionally define that the recipient may pass on the privilege to another user.

Only remove a DB2-PRIVILEGE member from the repository if you have generated an SQL REVOKE statement and do not need the member any more. If you need to grant the privilege again, retain the member, so that the repository reflects your long-term security arrangements.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-PRIVILEGE or SQL-PRIVILEGE member definitions using an AS clause.

Defining the Grantor of a Privilege

To define the user who is granting the privilege, enter:

```
GRANTOR user
```

where *user* is the name of a DB2-USER or SQL-USER member and represents the authorization ID of the user who is granting the privilege. The grantor is usually a database administrator for a project.

On encoding the GRANTOR clause is checked to ensure that the member named is a DB2-USER or SQL-USER member. The clause is not used in the generation of SQL statements and is for documentation purposes only.

Defining Database Privileges

To define privileges on databases, enter:

```
DATABASE
```

followed by one, several or all of these optional keywords that correspond to privileges allowed in DB2:

```
DBADM
DBCTRL
DBMAINT
CREATETAB
CREATTS
DISPLAYDB
DROP
IMAGCOPY
LOAD
RECOVERDB
REORG
REPAIR
STARTDB
STOPDB
STATS
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning.

On encoding, the DATABASE clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-DATABASE to which the privilege applies must be defined using an ON clause.

Defining Plan Privileges

To define privileges on plans, enter:

```
PLAN
```

followed by one or both of the following privileges:

```
BIND
EXECUTE
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning.

On encoding, the PLAN clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-PLAN to which the privilege applies must be defined using an ON clause.

Defining System Privileges

To define privileges to monitor and maintain the system, enter:

```
SYSTEM
```

followed by one, several or all of the following privileges:

```
SYSADM
SYSOPR
BINDADD
BSDS
CREATEALIAS
CREATEDBA
CREATEDBC
CREATEESG
CREATETMTAB
DISPLAY
RECOVER
STOPALL
STOSPACE
TRACE
MONITOR1
MONITOR2
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning.

On encoding, the SYSTEM clause is checked to ensure that the privileges specified are valid.

Defining Table or View Privileges

To grant privileges on a table or view, enter:

```
TABLE
```

followed by one, several or all of the following privileges:

```
ALTER
DELETE
INDEX
INSERT
SELECT
REFERENCES
UPDATE
```

The privileges are identified by the same keywords as those defined in the IBM documentation, and have the same meaning.

To optionally define the column that the REFERENCES privilege applies to in the table or view, enter:

```
COLUMNS column-name
```

where *column-name* is the unqualified name of a column in a table that is identified in the ON clause.

To optionally define the columns that the UPDATE privilege applies to in the table or view, enter:

```
UPDATE column-definition
```

where *column-definition* is either:

(*integer*) *member* KNOWN-AS *local-name*, or:

group EXPAND

integer is the number of columns to be generated from the member

member is the name of an ITEM or GROUP member

local-name is the name of the column

group is the name of a GROUP member.

Refer to the documentation of the DB2-TABLE member type for details of defining columns.

On encoding, the TABLE clause is checked to ensure that the privileges specified are valid.

For the successful generation of SQL statements, the DB2-TABLE or DB2-VIEW to which the privilege applies must be defined using an ON clause.

Defining Collection Privileges

To grant privileges on a collection, enter:

```
COLLECTION
```

followed by one or both of the following privileges:

```
CREATE
PACKADM
```

Defining ALL Table or View Privileges

To grant all the privileges on a table or view, enter:

```
ALL
```

For the successful generation of SQL statements, the DB2-TABLE or DB2-VIEW to which all the table privileges apply must be defined using an ON clause.

Defining Use Privileges

To specify the privilege to use buffer pools, enter:

```
USE BUFFERPOOL bufferpool-name
```

where *bufferpool-name* is one or more of the following buffer pools, separated by commas: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BP10, BP11l, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

To specify the privilege to use all buffer pools, enter:

```
USE BUFFERPOOL ALL
```

To specify the privilege to use named storage groups, enter:

```
USE STOGROUP stogroup-name
```

where *stogroup-name* is the name of one or more DB2-STOGROUP members, separated by commas.

To specify the privilege to use named table spaces, enter:

```
USE TABLESPACE tbspace-name
```

where *tbspace-name* is the name of one or more DB2-TBSPACE members, separated by commas.

To specify the privilege to use distinct types (user-defined data types), enter:

```
USE DISTINCT-TYPE distinct-type
```

where *distinct-type* is the name of an ITEM member with the DISTINCT-TYPE attribute that defines the data type.

To specify the privilege to use JARs (JAVA archive), enter:

```
USE JAR  jar-name
```

where *jar-name* is the fully qualified name of the JAVA archive file.

Note:

You can specify only one USE clause.

On encoding, the USE clause is checked to ensure that the privilege specified is valid.

Defining the Object to Which the Privilege Applies

To define the object to which the privilege applies, enter:

```
ON member
```

where *member* is the name of one of these members:

- DB2-DATABASE for database privileges
- DB2-PLAN for plan privileges
- DB2-TABLE, DB2-VIEW, SQL-TABLE, or SQL-VIEW for table and all table privileges.

This clause must be present for the successful generation of SQL statements.

Defining Recipients of a Privilege

To define the user to whom the privilege applies, enter:

```
TO user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the user.

To define that the privilege applies to all users on the local subsystem, enter:

```
TO PUBLIC
```

To define that the privilege applies to all users on the local and global system, enter:

```
TO PUBLIC AT ALL LOCATIONS
```

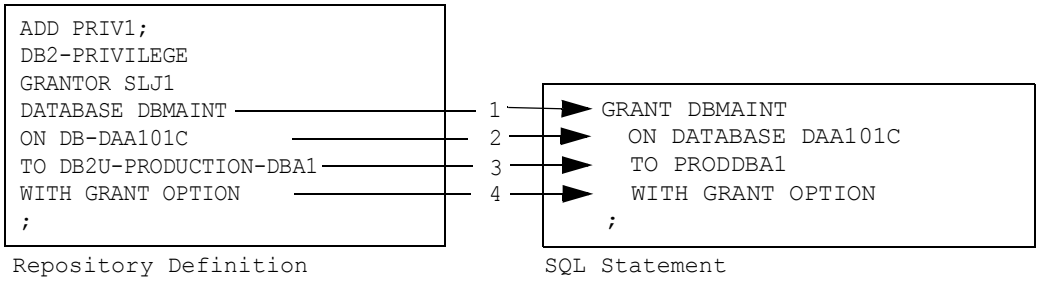
If you want to interrogate the repository to find out which privileges are granted to all users, define a DB2-USER member named PUBLIC.

To optionally specify that the recipient(s) of the privilege may transfer it to another user enter:

```
WITH-GRANT-OPTION
```

The TO clause must be present for the successful generation of an SQL statement.

Example of a DB2-PRIVILEGE Definition for a Database, and Generated SQL Statement



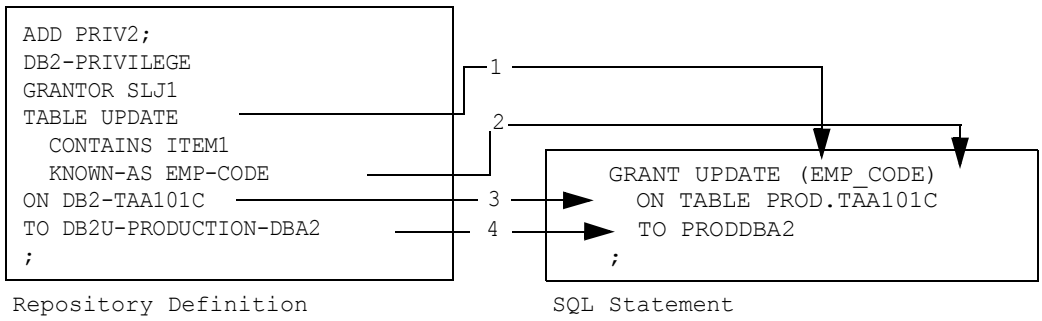
1 The type of privilege being granted is taken directly from the member definition.

2 The DB2 name for the database on which the privilege is being granted is derived from the SQL ALIAS defined in the DB2-DATABASE member.

3 The DB2 name of the user to whom the privilege is being granted is derived from the SQL ALIAS of the DB2-USER member, DB2U-PRODUCTION- DBA1.

4 WITH GRANT OPTION is derived from the WITH-GRANT-OPTION keyword in the member definition.

Example of a DB2-PRIVILEGE Definition for a Table and Generated SQL Statement



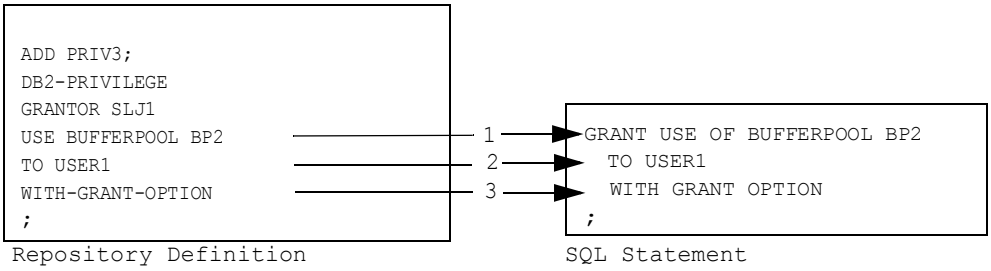
1 The type of privilege being granted is taken directly from the member definition.

2 The DB2 name for the column on which the update privilege is granted is taken from the local name, defined in the KNOWN-AS clause.

3 The DB2 name of the table on which the privilege is being granted is derived from the SQL ALIAS defined in the DB2-TABLE, qualified by the SQL ALIAS of the DB2-USER member defined as the CREATOR-OWNER of the DB2-TABLE.

4 The DB2 name of the user to whom the privilege is being granted is derived from the SQL ALIAS of the member DB2U-PRODUCTION-DBA2.

Example of a DB2-PRIVILEGE Definition for a Buffer Pool and Generated SQL Statement



1 The type of privilege being granted is taken directly from the member definition.

2 The DB2 name of the user to whom the privilege is being granted is taken directly from the name of the member USER1.

3 WITH GRANT OPTION is derived from the WITH-GRANT-OPTION keyword in the member definition.

DB2-PRIVILEGE Syntax

```
>>-- DB2-PRIVILEGE --+-------------+--+-------------+-->
                     |  AS privilege |  |GRANTOR user |
                     +-------------+  +-------------+

>--+-------------------------------------------------------------+-->
   |                                   <<<<<,<<<<                |
   |--DATABASE database privilege------ON database               |
   |                                   <<<,<<<                   |
   |--PLAN--+-BIND-----+---------------ON plan                   |
   |        |-EXECUTE--|                                         |
   |                                   <<<<<,<<<<                |
   |--TABLE table-privilege------------ON--+-table-+             |
   |  ALL                                  |-view--|             |
   |                                   <<<<<<<,<<<<<             |
   |--COLLECTION--+-CREATE-----+-------ON collection             |
   |              |-PACKADM----|                                  |
   |--PACKAGE-----+-ALL--------+-------ON package                |
   |              |-BIND-------|                                  |
   |              |-COPY       |                                  |
   |              |-EXECUTE    |                                  |
   |              |-RUN        |                                  |
   |--SYSTEM system-privilege----------------------------------- |
   |--USE use-privileges---------------------------------------- |
   +-------------------------------------------------------------+

>--+-------------------------------------------------------------+-->
   |--TO--+-<< , <<-----------+-------------+-----+-------------+|
   |      |-user-------------|             |     |             |
   |      |-PUBLIC-----------+-------------+     |             |
   |                         |-AT-ALL-LOCATIONS-|              |
   +-------------------------------------------------------------+

>--+-------------+--+-----+-->
   |-common clauses-|  |-;-|
   |                |  |-.-|
```

where:

privilege is the name of a DB2-PRIVILEGE or SQL-PRIVILEGE member

user is the name of a DB2-USER or SQL-USER member.

database-privileges are:

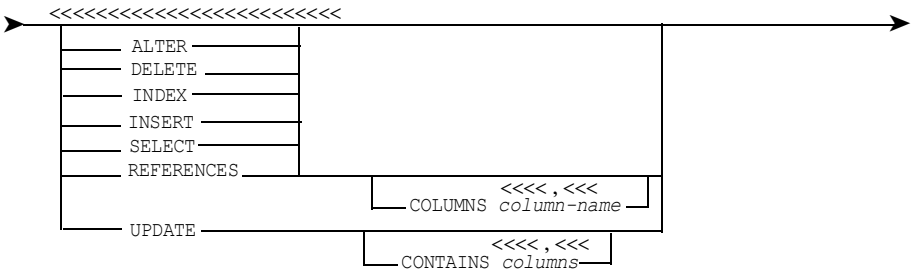
```
>>--<<<<<
```

where:

database is the name of a DB2-DATABASE member

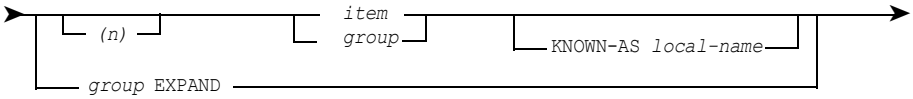
plan is the name of a DB2-PLAN member.

table-privileges are:



where *column-name* is the name of a column in a table identified in the ON clause.

where *columns* are:



where:

n is the number of columns in a column set.

item is the name of an ITEM member.

group is the name of a GROUP member.

local-name is a name, of no more than 18 characters.

table is the name of a DB2-TABLE or SQL-TABLE member.

view is the name of a DB2-VIEW or SQL-VIEW member.

collection is the name of a DB2-COLLECTION member.

package is the name of a DB2-PACKAGE member.

system-privileges are:

```
<<<<<<<<<<<<<<<<<<<<<<<<<<
ARCHIVE
BINDAGENT
SYSCTRL
SYSADM
SYSOPR
BINDADD
BSDS
CREATEALIAS
CREATEDBA
CREATEDBC
CREATESG
CREATETMTAB
DISPLAY
RECOVER
STOPALL
STOSPACE
TRACE
MONITOR1
MONITOR2
```

use-of-privileges are:

```
                                  <<<<,<<<<
BUFFERPOOL                          bpname
                                    ALL
STOGROUP stogroup-name
TABLESPACE tbspace-name
DISTINCT-TYPE item-name
    <<<<,<<<<
JAR jar-name
```

where:

DISTINCT-TYPE *item-name* defines a particular datatype, where *item-name* is the name of an ITEM member with the desired datatype.

jar-name is a fully qualified Java Archive file name.

bpname is BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BP10, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP4O, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48B, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

stogroup-name is the name of a DB2-STOGROUP member.

tbspace-name is the name of a DB2-TBSPACE member.

user is as defined above.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-PROCEDURE

DB2-PROCEDURE defines DB2 procedures and functions.

In the DB2 environment, *stored procedures* are kept in the DB2 catalog and can be called by applications or activated by triggers.

Refer to "Reusing Existing Member Definitions" on page 509 for details on reusing all or part of existing DB2-PROCEDURE member definitions using an AS clause.

Defining a Procedure

To define a DB2 procedure, enter:

```
DB2-PROCEDURE
```

Followed by the attributes for this procedure.

Defining an Owner

To define the owner of this function, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the procedure. This clause is required for the successful generation of SQL statements, but can be overridden by an SQLID clause in the DB2 export command used.

Defining a Comment

To define a comment for this procedure, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is the comment for this procedure. This value is used to generate the SQL COMMENT ON statement when executing an export command.

Defining Parameters

To define parameters consisting of a list of items that are passed to this procedure, enter:

```
PARAMETERS parm-1, parm-2,…
```

where *parm-1* and *parm-2* are names of ITEM, DB2-TABLE, or DB2-VIEW members which describe the parameters being passed.

You also enter these additional attributes:

- CCSID value
- Whether the parameter is an input, output, or both
- Whether the parameter is a pointer to a LOB type item (AS LOCATOR)

Defining the LUNAME and LOADMOD Name

For DB2 Version 5.0 and earlier, the LUNAME was acquired when procedure and procedure definitions were imported from DB2. The DB2 catalog table SYSIBM.SYSPROCEDURES which contains this attribute is not used in later releases of DB2. For this reason, LUNAME may not apply for your installation.

To define an LUNAME of the system that issued the SQL CALL statement to invoke this procedure, enter:

```
LUNAME name
```

where *name* is the luname of the system.

To define a LOADMOD name of the MVS load module that DB2 should load to satisfy the request for the stored procedure, enter:

```
LOADMOD name
```

where *name* is the 8 character member name of the MVS load module.

Defining the LINKAGE Attribute

To define the linkage convention used to pass parameters to the procedure, enter:

```
LINKAGE NULL
```

when the SIMPLE WITH NULLS convention is used where an indicator array is passed to the stored procedure. Null input parameters are allowed.

Enter:

```
LINKAGE BLANK
```

where SIMPLE linkage convention is used where input parameters cannot be null.

Specifying Compatible Attribute Values

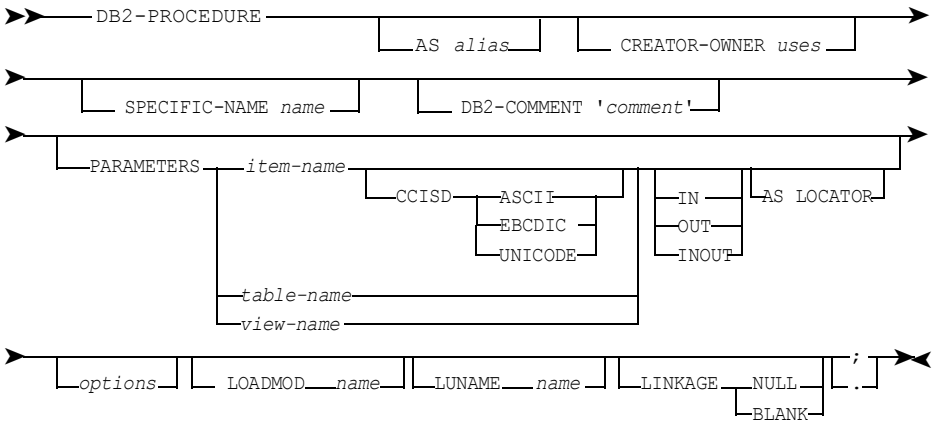
All the rules for consistent attribute values that apply in DB2 also apply to the definition of DB2-PROCEDURE members, but they are only enforced when an export command such as DB2 CREATE is issued for the member. For example, when LANGUAGE JAVA is specified, DB2 requires that the PARAMETER-STYLE must also be JAVA. A procedure may be defined with LANGUAGE JAVA that does not specify PARAMETER-STYLE JAVA, but this error is flagged when the procedure is used to generate a SQL statement to export this definition to DB2.

These are the current attribute dependencies:

- When LANGUAGE is JAVA, the following apply:
  - PARAMETER-STYLE must be JAVA
  - EXTERNAL-NAME must be specified
  - FINAL-CALL NO must be specified
  - DBINFO NO must be specified
- Do not specify EXECUTES UPDATE-SQL when ALLOW-PARALLEL is specified.

There are many similarities between the DB2-FUNCTION and DB2-PROCEDURE member types. The main difference is the presence of the RETURNS clause in the DB2-FUNCTION definition. For this reason, a DB2-FUNCTION that is defined without a RETURNS clause is treated as a DB2-PROCEDURE member type when an export command is issued for it.

DB2-PROCEDURE Syntax



where:

`DB2-COMMENT 'comment'` specifies the comment for this procedure, and is a string of up to 254 characters.

table-name is the name of a DB2-TABLE that describes the parameters passed to this procedure.

view-name is the name of a DB2-VIEW that describes the parameters passed to this procedure.

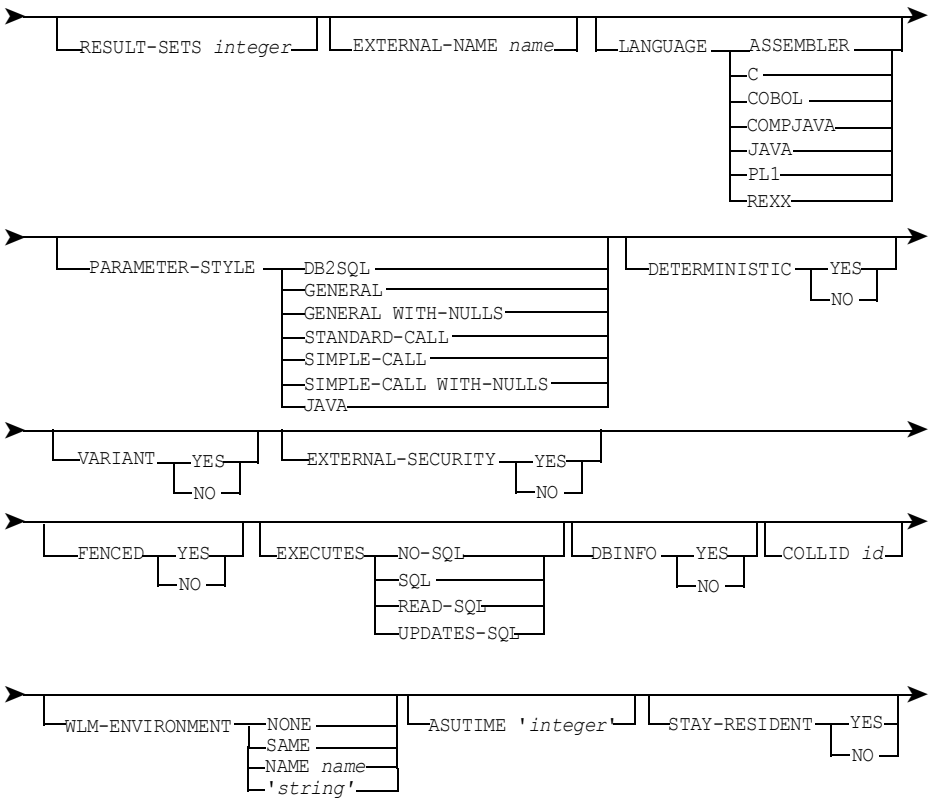
alias is the name of a DB2-PROCEDURE member.

uses is the name of a DB2-USER member.

parm-name is the name of an ITEM member.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of common clauses.

options is:



```
──┬──────────────
```

When LANGUAGE is JAVA, the following apply:

- PARAMETER-STYLE must be JAVA
- EXTERNAL-NAME must be specified
- DBINFO NO must be used
- WLM-ENVIRONMENT must be defined
- RUN-OPTIONS must not be used

When LANGUAGE is COMPJAVA, EXTERNAL-NAME is required.

When LANGUAGE is REXX, the following apply:

- PARAMETER-STYLE GENERAL or GENERAL WITH-NULLS should be used
- WLM-ENVIRONMENT must be defined
- PROGRAM-TYPE SUB must be used

Do not specify WLM-ENVIRONMENT when PROGRAM-TYPE SUB is specified.

The following are synonyms:

- PARAMETER-STYLE STANDARD-CALL is DB2SQL
- PARAMETER-STYLE SIMPLE-CALL is GENERAL
- VARIANT is NOT DETERMINISTIC
- VARIANT NOT is DETERMINISTIC
- NULL-CALL is CALLED-ON-NULL-INPUT

Synonyms include STANDARD-CALL for DB2SQL, SIMPLE-CALL for GENERAL, and SIMPLE-CALL WITH-NULLS for GENERAL WITH-NULLS.

DB2-STOGROUP

DB2-STOGROUP defines DB2 storage groups in the repository.

Refer to "DB2-STOGROUP Syntax" on page 424 for the syntax of the DB2-STOGROUP member definition.

To define a DB2 storage group, enter:

```
DB2-STOGROUP
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-STOGROUP members are optional. However, the VOLUMES and VCAT clauses must be present for the successful generation of SQL CREATE STOGROUP statements.

Refer to Chapter 4, "Repository Definition," on page 91 for details of generating DB2 storage group names.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-STOGROUP member definitions using an AS clause.

Defining the DASD Volumes on which a Storage Group Resides

To define the DASD volumes on which the storage group resides, enter:

```
VOLUMES vol-id-list
```

where *vol-id-list* is one or more storage volume names, each of no more than six characters, and separated by commas. You can define a maximum of 133 storage volume names.

At least one volume must be defined for the successful generation of an SQL CREATE STOGROUP statement.

When you generate SQL statements, DB2 names for volumes are taken directly from the names you specify in this clause.

You can arrange for the Storage Management Subsystem (SMS) to manage the storage needed for the storage group supports. To do this, enter:

```
VOLUMES *
```

Defining Storage Space

To define the VSAM catalog the storage group is to use, enter:

```
VCAT catalog
```

where *catalog* is the name of a VSAM catalog, of no more than eight characters.

When you generate SQL statements, VCAT catalog names are taken directly from the name you specify in this clause. The VCAT clause must be defined for the successful generation of SQL CREATE STOGROUP statements.

Defining a VSAM Catalog Password

To define a control or master level password used to access the VSAM catalog, enter:

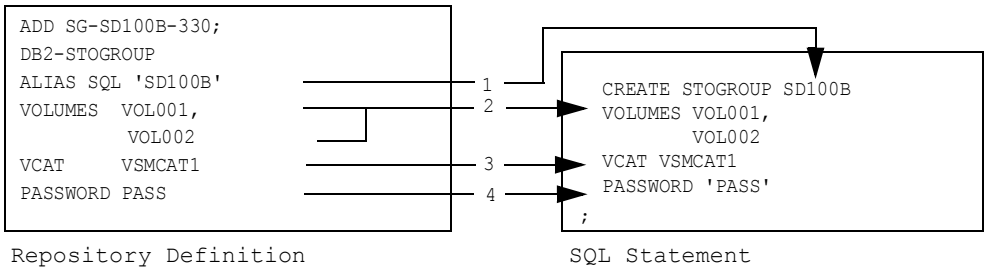
```
PASSWORD password
```

where *password* is a control or master level VSAM catalog password, of no more than 8 characters.

When you generate SQL statements, the VCAT password is taken directly from the password you specify in this clause.

Example

The DB2-STOGROUP member SG-SDl00B-300 is defined in the repository using an add command. An SQL statement is generated from the member definition using a DB2 BIND command.



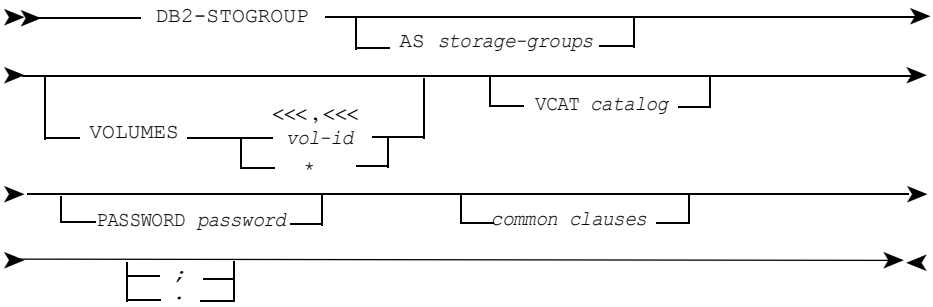
1 The storage group name SD100B is taken from the SQL ALIAS.

2 The volume names are taken from the member definition.

3 The VSAM catalog name is taken from the member definition.

4 The password is taken from the member definition. Delimiters are generated to allow inclusion of special characters.

DB2-STOGROUP Syntax



where:

storage-groups is the name of a DB2-STOGROUP member.

vol-id is a storage volume name, of no more than 6 characters.

catalog is a VSAM catalog name, of no more than 8 characters.

password is a VSAM catalog password, of no more than 8 characters.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of `common clauses`.

DB2-TABLE

DB2-TABLE defines DB2 tables in the repository.

Refer to "DB2-TABLE Syntax" on page 450 for the syntax of the DB2-TABLE member definition.

Tables are of major interest to the end-user, since they contain the user's own data and are not a product of the database administrator or other technical specialist. Therefore the DB2-TABLE is one of the most used DB2 member types.

To define a DB2 table, enter:

```
DB2-TABLE
```

The member definition must begin with this member type identifier.

All other clauses available to define DB2-TABLE members are optional. However, for the successful generation of SQL statements you must define specific clauses, as follows:

- For ALTER TABLE statement define the CREATOR-OWNER clause
- For CREATE TABLE statements define the CREATOR-OWNER, COLUMNS and IN clauses
- For COMMENT ON statements define the CREATOR-OWNER and DB2-COMMENT clauses
- For DECLARE TABLE statements define the CREATOR-OWNER and COLUMNS clauses
- For DROP TABLE statements define the CREATOR-OWNER clause
- For LABEL ON statements define the CREATOR-OWNER and DB2- LABEL clauses.

To specify the ITEM and GROUP members that represent the columns of the table, use the CONTAINS clause. It establishes relationships between a DB2-TABLE and ITEM and GROUP members, which define the table's columns. These GROUPs and ITEMs may also form part of other file and database segment definitions. For example, installations with IMS may already have GROUP and ITEM definitions in the repository, which can now be shared with the DB2 environment.

You can define columns:

- Individually, so that one ITEM or GROUP member defines one column
- In sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- In cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column.

Sub-clauses within the COLUMNS clause enable you to define:

- The names of columns
- Whether or not the column can contain a null value
- The attributes of the column, for example, if it is a primary key, or if it is to have an associated comment
- Referential constraints on the table.

Generic clauses enable you to define extra column attributes for relational tables other than DB2 tables.

The DB2 facilities FIELDPROC, EDITPROC, VALIDPROC, AUDIT, COMMENT, and LABEL are supported by the FIELDPROC, EDITPROC, VALIDPROC, AUDIT, DB2-COMMENT, and DB2-LABEL clauses respectively.

You can specify the number of rows in a table using the CARDINALITY clause. This enables you to calculate the minimum and maximum size of the table.

DB2-TABLE repository definitions can be generated automatically if you use the Workbench Design Area (WBDA) facilities for DB2 database design.

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of generating DB2-TABLE member definitions from the WBDA.

Refer to Chapter 4, "Repository Definition," on page 91 for details of the derivation of DB2 table names.

Refer to Appendix A, "Name Reduction Process," on page 511 for details of generic relational support.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-TABLE or SQL-TABLE member definitions using an AS clause.

Defining an Owner

To define the owner of a table, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the table.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

This clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Note:

The DB2-USER or SQL-USER member named in the CREATOR-OWNER clause is used to generate user-qualified names for tables. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated. The location-qualifier can be overridden by a LOCATION clause defined in a DB2 command.

Defining the Table Space in Which a Table is Created

To define the table space in which the table is created, enter:

```
IN tbspace
```

where *tbspace* is the name of a DB2-TBSPACE member.

On encoding, the member specified is checked to ensure it is a DB2-TBSPACE member.

For the successful generation of SQL statements the DB2-TABLE definition must include an IN clause, naming a DB2-TBSPACE member. The DB2-TBSPACE must include an IN clause, naming a DB2-DATABASE. This chain of relationships defines the database to which the table space and table belong.

Temporary

A table described by the SQL statement CREATE GLOBAL TEMPORARY TABLE and used to hold data temporarily, such as the intermediate results of SQL transactions. Temporary tables persist as long as the application supports them.

Temporary tables should be defined on the repository as a DB2-TEMPTABLE member type. However, it can also be defined as a DB2-TABLE member type with the TEMPORARY attribute. DB2-TABLE is the base member type for DB2-TEMPTABLE. When a temporary table is defined as a DB2-TABLE member type, however, listing only temporary tables by member type alone is not possible.

Auxiliary

For every LOB column of a table there must be an AUXILIARY TABLE. The STORES attribute names the table containing the LOB column. The COLUMN CONTAINS attribute specifies the name of this LOB column.

Auxiliary tables should be defined on the repository as a DB2-AUXTABLE member type. However, it can also be defined as a DB2-TABLE member type with the AUXILIARY attribute. DB2-TABLE is the base member type for DB2-AUXTABLE. When an auxiliary table is defined as a DB2-TABLE member type, however, listing only auxiliary tables by member type alone is not possible.

Reusing Existing Column Structures

In DB2, you may wish to create a table with the same column structure as an existing table, so that you can use it in production and development. To copy the columns of an existing table in DB2, enter:

```
LIKE member
```

where *member* is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW, or SQL-VIEW member.

When you generate SQL CREATE TABLE statements from a DB2-TABLE that includes a LIKE clause, the statement also contains a LIKE clause. When the statement is submitted to DB2, the table or view named in the LIKE clause must exist, or the column structure cannot be copied.

If you want to generate SQL DECLARE statements or host language data structures from a DB2-TABLE defined using a LIKE clause, you must also define an AS clause referring to the same member.

For example:

```
LIKE TA-TOTAL-STOCK
AS TA-TOTAL-STOCK
```

defines that the table copies the generated columns of the DB2-TABLE TA-TOTAL-STOCK.

Defining a Comment on a Table or Column

To define a comment for a table or column, enter:

```
keyword 'comment'
```

where *keyword* is one of these:

- DB2-COMMENT for DB2 tables and columns
- COL-COMMENT for any column

comment is a string of no more than 254 characters, each line of which is within delimiters.

When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space between the last character of each continuing line and its delimiter.

For example, the DB2-TABLE named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

```
DB2-COMMENT 'This table contains the Manager number of every '
'manager in each department'
```

The following SQL statement can be generated:

```
COMMENT ON TABLE PERSONNEL.MANAGER_NUMBER IS 'This table cont
ains the Manager number of every manager in each department'
```

In this example the word *contain* has been split due to the margins set in the DB2 profile.

The DB2-COMMENT or COL-COMMENT clause must be present for the successful generation of SQL COMMENT ON statements.

Note:

- For columns, the comment definition must be part of the CONTAINS clause defining that column or group of columns
- For tables, the comment definition should precede the COLUMNS clause that defines the columns of the table.

Defining a Label on a Table or Column

To define a label for a table or column, enter:

```
keyword 'label'
```

where *keyword* is one of these:

- DB2-LABEL for DB2 tables and columns
- COL-LABEL for any column

label is a string of no more than 30 characters, within delimiters.

The DB2-LABEL or COL-LABEL clause must be present for the successful generation of SQL LABEL ON statements.

Note:

- For columns, the label definition must be part of the CONTAINS clause defining that column or group of columns
- For tables, the label definition should precede the COLUMNS clause that defines the columns of the table.

Specifying the Form Description that Defines the Data Type of Columns

To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns in the table, enter:

```
COLUMNS form-keyword
```

where *form-keyword* is one of the following:

- ENTERED-AS
- HELD-AS
- REPORTED-AS
- DEFAULTED-AS

The form keyword that you define in the COLUMNS clause applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

For example, a DB2-TABLE member containing the following lines:

```
COLUMNS ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the 2 ITEM members:

```
ITEM1                        ITEM2

ITEM                         ITEM
HELD-AS BINARY 10            ENTERED-AS DECIMAL 4.2
ENTERED-AS CHAR 5            DEFAULTED-AS FLOAT 7
REPORTED-AS FLOAT 9
```

The ENTERED-AS form keyword in the DB2-ABLE definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns. Therefore the column generated from ITEM1 has a data type of CHAR, and the column generated from ITEM2 has a data type of DECIMAL. If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the Manager Products defaults apply. For further information refer to the *ASG-Manager Products Source Language Generation* guide.

Refer to Appendix C, "Defining and Generating DB2 Member Types," on page 519 for further details of documenting the columns of tables and generating data types.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of form keywords and form description in ITEM and GROUP member definitions.

Specifying the ITEMs or GROUPs that Define Columns

To specify the ITEM or GROUP members that define columns, enter:

`CONTAINS` *`member-list`*

where *`member-list`* is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either ITEMs or GROUPs. Duplicate column names are not permitted by DB2, therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

`CONTAINS` *`item version`*

where:

`item` is the name of an ITEM member

`version` is an integer in the range 1 to 15, and defines the form description version that you want to use.

For example:

```
COLUMNS HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description in the ITEM member STOCK-LIST is used as the column data type.

When you use the SIZE and RECALCULATE commands, the data type of columns is used to calculate the size of a table.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

where:

integer is the number of columns to be derived from the member, within brackets

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by _1, the second by_2 an so on.

For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST_1, STOCK_LIST_2, STOCK_LIST_3, and STOCK_LIST_4. The attributes, such as data type, are the same for each of the four columns.

When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where a DB2 command applied to the DB2-TABLE specifies the EXPAND keyword, then each ITEM within a GROUP generates a separate column.

Naming Columns

You can explicitly name a column if you do not want its name to be derived from the ITEM or GROUP name or alias.

To define the name of a column in a table, enter:

```
KNOWN-AS local-name
```

where *local-name* is a string of no more than 18 characters.

For example:

```
CONTAINS IT-INCOMING KNOWN-AS STOCK-IN
```

defines that the ITEM member IT-INCOMING generates a table column called STOCK_IN.

If you use the KNOWN-AS clause to name a set of columns, the local name is duplicated for each column. To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by _1, the second by _2 and so on.

For example:

```
CONTAINS (3) IT-Q1 KNOWN-AS MONTH
```

generates three columns from the ITEM member IT-Q1 named MONTH_1, MONTH_2, and MONTH_3.

Refer to Chapter 4, "Repository Definition," on page 91 for further details of the generation of column names.

Specifying that Each ITEM Contained in a GROUP Defines One Column

If you want each of the ITEMs contained in a GROUP to represent a column, enter:

```
CONTAINS group EXPAND
```

where *group* is the name of a GROUP member.

For example:

```
CONTAINS AREA-DEPOT
```

generates four columns because the GROUP, AREA-DEPOT, contains four ITEMs.

However, if the NO-EXPAND keyword is specified in a DB2 command, the EXPAND keyword in the member definition is overridden and the GROUP generates a single column.

Note:

You cannot define a KNOWN-AS clause with expanded GROUPs. The generation of SQL CREATE statements is unsuccessful if a DB2 command including the EXPAND keyword is applied to a DB2-TABLE whose columns are named by the KNOWN-AS clause. This is because the local name is duplicated for every column generated from the GROUP member.

If the GROUP is nested, that is it contains other GROUPs, each of these is also expanded so that all ITEMs are used to define columns. Nesting can continue to any depth and is only limited by the amount of memory available.

ELSE clauses defined in expanded GROUP members generate a column with a CHAR or VARCHAR data type with a field length equal to the longest overlaid field.

Where a set of columns is derived from an expanded GROUP, each contained ITEM or GROUP is repeated the number of times specified. If a GROUP contains an ITEM with its own repeating factor, the ITEM is also repeated the number of times specified.

For example, a DB2-TABLE defined as:

```
CONTAINS (2) AREA-DEPOT EXPAND
```

where the GROUP, AREA-DEPOT, contains:

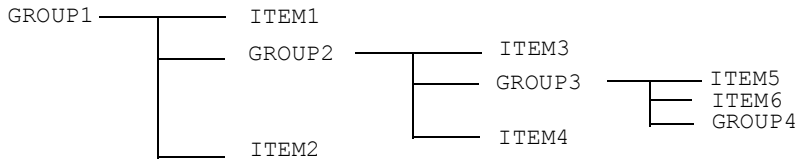
```
(2)      WAREHSE-A
         WAREHSE-B
(3)      LOCALSTK-A
```

generates four columns from WAREHSE-A, two columns from WAREHSE-B, and six columns from LOCALSTK-A, and names them as follows:

```
WAREHSE_A_1
WAREHSE_A_2
WAREHSE_B_1
LOCALSTK_A_1
LOCALSTK_A_2
LOCALSTK_A_3
WAREHSE_A_3
WAREHSE_A_4
WAREHSE_B_2
LOCALSTK_A_4
LOCALSTK_A_5
LOCAKSTK_A_6
```

Expanded GROUPs and Host Language Data Structures

The member GROUP1 contains nested GROUPs, shown in the following diagram.



When you generate SQL statements or SQL host language data structures, intermediate levels in the data structure, that is GROUP2 and GROUP3, are removed in order to generate the following flat, two-level structure:

```
02  ITEM1
02  ITEM3
02  ITEM5
02  ITEM6
02  GROUP4
02  ITEM4
02  ITEM2
```

Note:

GROUP4 is treated as an elementary field as it has no lower level. Its data type defaults to CHAR(l).

Intermediate levels, in the above example GROUP2 and GROUP3, can be shown as comments.

When you generate host language data structures in working storage, for example for PL/1, intermediate levels are expanded to give the following nested structure:

```
02 ITEM1
 02 GROUP2
    03 ITEM3
      03 GROUP3
         04 ITEM5
         04 ITEM6
         04 GROUP4
     03 ITEM4
02 ITEM2
```

If you consider the original nested structure as a tree and fields that do not have lower levels as leaves, then the root of the tree is taken as the first level and only the leaves of the tree are taken as the second level.

There is one exception to this rule: when COBOL data structures are generated, VARCHAR and VARGRAPHIC characters are not treated in this way, since they have a two-level structure anyway.

Refer to the PRODUCE command for details of generating host language data structures.

Defining Column Attributes

You can define additional attributes for a table's column(s) using sub-clauses within the CONTAINS clause. If the CONTAINS clause defines a column set or an expanded GROUP, the generated column attributes apply to all the columns in the set.

To specify the contents of the column as single-byte data, enter:

```
FOR-SBCS-DATA
```

To specify the contents of the column as mixed data, enter:

```
FOR-MIXED-DATA
```

To specify the contents of the column as bit (binary) data, enter:

```
FOR-BIT-DATA
```

Note:

The FOR-BIT-DATA keyword is only valid where columns have the data-type of CHAR, VARCHAR, or LONG VARCHAR specified in the form description.

To define that columns cannot contain a null value, and are set to a default value by DB2, enter:

```
WITH-DEFAULT
```

The WITH-DEFAULT keyword generates NOT NULL WITH DEFAULT in SQL statements.

To specify that a column cannot contain a null value, enter:

```
NOT-NULL
```

NOT-NULL and WITH-DEFAULT are mutually exclusive. If you specify both, the member will not encode.

Following WITH-DEFAULT you may specify a default value to be assigned to the column. This may be a constant or any of the literals USER, CURRENT-SQLID, or NULL. You may also specify a cast-function if the column is defined as a distinct type column.

The CAST attribute specifies the name of a cast function that matches the name of an ITEM member that defines a distinct type for the column. A cast function can only be specified if the column is a distinct type.

GENERATED is only allowed if the column is defined by an ITEM of the data type ROWID or is defined as an identity column. Sub-attributes of GENERATED are:

- ALWAYS specifies that DB2 always generates a value for this column when inserting or loading this table.
- BY-DEFAULT specifies that DB2 will generate a value for the column when a row is inserted into the table unless a value is specified.
- AS-IDENTITY specifies that the column is an identity column for the table. A table can have only one identity column. AS IDENTITY can be specified only if the data type for the column is an integer with no decimal positions. An identity column is implicitly NOT-NULL. The sub-attributes of the AS-IDENTITY clause are:

  — START-WITH *numeric-constant*

  Specifies the first value for the identity column, and can be positive or negative.

  — INCREMENT-BY numeric-constant

  Specifies the interval between consecutive values of the identity column, and can be positive or negative.

  — CACHE integer

  Specifies the number of values of the identity column sequence that DB2 preallocates and keeps in memory. The default is CACHE 20.

  — NO CACHE

  Specifies that values for the identity column are not preallocated.

  — CYCLE

  Specifies that values continue to be generated for this column after the maximum or minimum value has been reached.

  — NO CYCLE

  Specifies that values will not be generated for the identity column once the maximum or minimum value for the sequence has been reached. This is the default.

  — MAXVALUE *numeric-constant*

  Specifies the numeric constant that is the maximum value that is generated for this identity column.

  — MINVALUE *numeric-constant*

  Specifies the numeric constant that is the minimum value that is generated for this identity column. This value must be less than the maximum value.

To specify a check constraint on a column, to specify which values of the column are valid, enter:

```
CHECK-CONSTRAINT NAMED constraint-name CONDITION  'check-condition'
```

where:

check-condition is an expression up to 255 bytes long and may be repeated as many times as required, as long as the CONDITION keyword is also repeated (no comma)

constraint-name is an SQL long identifier of up to 18 characters.

You may apply a check constraint to one or more columns in a table when it is created or altered.

Columns with a data type of CHAR can also have an associated FIELDPROC procedure, provided WITH-DEFAULT is not defined. To define a FIELDPROC, enter:

```
FIELDPROC process
```

where *process* is the name of a SYSTEM, MMR-SYSTEM, PROGRAM, or MODULE member, and represents a field procedure that exists in DB2.

To define the parameters passed to the field procedure, enter:

```
CONSTANT list
```

where *list* is one or more parameters separated by commas. The list must be no more than 254 characters, within delimiters.

For example:

```
CONTAINS IT-INCOMING NOT-NULL FIELDPROC MOD-XT3 CONSTANT
"CONST4-3, CONST4-4"
```

defines that the ITEM member IT-INCOMING generates a column that:

- Cannot contain a null value, and
- Uses the process member MOD-XT3, passing the parameters CONST4-3 and CONST4-4.

To define that a column forms the primary key of a table, enter:

```
PRIMARY-KEY
```

A primary key can comprise a maximum of 16 columns. For the successful generation of an SQL CREATE TABLE statement, columns defined as PRIMARY-KEY must also be defined as either NOT-NULL or WITH- DEFAULT.

Where your primary key consists of several columns, the sequence in which columns constitute the key is assumed to be the sequence in which they are defined, unless you specify the key position.

To define the position of the column in a multi-column primary-key, enter:

```
PRIMARY-KEY kpos
```

where *kpos* is an integer in the range 1 to 16

For example:

```
CONTAINS
     DEPT-NO
    ,EMP-NO NO-NULL PRIMARY-KEY 2
    ,JOB-TITLE
    ,EMP-NAME NOT-NULL PRIMARY-KEY 1
```

defines that the primary key of this table is EMP-NAME followed by EMP-NO, rather than EMP-NO followed by EMP-NAME.

To define that the entries in a primary key column are sorted in ascending key order in the index, enter:

```
PRIMARY-KEY ASC
```

To define that the entries in a primary key column are sorted in descending key order in the index, enter:

```
PRIMARY-KEY DESC
```

To define that the column is contained in a unique key for a table, enter:

```
UNIQUE
```

A unique key of a table can contain up to 64 columns, whose total length must not exceed 255. Multiple unique keys can be defined to a table, and a column can be in multiple unique keys, but cannot be repeated within a single key, and no two unique keys may be identical.

When a unique key consists of multiple columns, the sequence in which the columns constitute the key is assumed to be the sequence in which they are defined as columns in the table, unless you specify a key position by entering:

```
UNIQUE kpos
```

where *kpos* is an integer in the range of 1 to 64.

When multiple unique keys are defined containing the current column, the unique keys must be identified by either its index name or its constraint name. To specify that the column is a column in a unique key with an index name, enter:

```
UNIQUE index-name kpos
```

where *index-name* is the DB2 index name of the unique key, and the optional kpos is the sequence number of the column's position within the key.

To specify that the column is a column in a unique key with a unique constraint name, enter:

```
UNIQUE UNIQUE-CONSTRAINT name kpos
```

where *name* is the DB2 unique constraint name of the unique key, and the optional *kpos* is the sequence number of the column's position within the key.

To define that a column is to contain a label enter:

```
DB2-LABEL 'label'
```

Defining Generic Column Attributes

If you use the DB2-TABLE member type to define other types of relational table, for example ORACLE or SQL/DS tables, additional column attributes are available.

To define the percentage of free space in a primary key, enter:

```
PCTFREE pn
```

where *pn* is an integer in the range 0 to 99.

To define column comments for a relational table, enter:

```
COL-COMMENT 'comment'
```

To define column labels for a relational table, enter:

```
COL-LABEL 'label'
```

COL-COMMENT and COL-LABEL are alternatives to DB2-COMMENT and DB2-LABEL, and follow the same rules.

To define a relationship between a member representing columns of a relational table and another repository member, enter:

```
column-relationship member
```

where:

column-relationship is COL-REL1, COL-REL2, or COL-REL3.

member is the name of a repository member.

To define additional attributes for columns in a relational table, enter:

column-attribute '*string*'

where:

column-attribute is COL-ATT1, COL-ATT2, or COL-ATT3.

string is a string of no more than 254 characters, within delimiters.

Note:

COL-COMMENT and COL-LABEL can be used to generate comments and labels for DB2 tables, but the other clauses are ignored by DB2 commands.

Refer to Appendix B, "Documenting Other Relational Databases," on page 515 for further details of defining repository members for other types of relational database.

Primary Key Constraint

A primary key is defined as a column attribute on the columns that comprise the key. The primary key constraint is specified separately after the column definitions.

To define a primary key constraint name, enter:

PRIMARY-KEY-CONSTRAINT *name*

where *name* is the constraint name of up to 128 characters.

Defining Referential Constraints

To define a referential constraint for a table, enter:

CONSTRAINT *constraint*

where *constraint* is one, several or all of these:

- The NAMED clause defines a constraint name
- The FOREIGN-KEY clause specifies one or more columns to form the foreign key
- The REFERENCES clause specifies the table being referenced
- The COLUMNS clause specifies the column being referenced
- The DELETE clause defines the associated DELETE rule.

You may define any number of referential constraints for a table. Each referential constraint requires its own CONSTRAINT clause. Each CONSTRAINT clause must include the FOREIGN-KEY and REFERENCES clauses for the successful generation of SQL CREATE statements.

You can name a referential constraint without specifying the column(s) that form the foreign key. Thus you can set up DB2-TABLE definitions with named referential constraints between them before deciding on the contents of the tables. This feature is useful in a top-down approach to database design.

To define a constraint name, enter:

```
NAMED constraint-name
```

where *constraint-name* is the name, of no more than 8 characters, by which the referential constraint is known to DB2.

Each constraint name must be unique within a table. If you do not specify a constraint name, a default name is generated by DB2.

To define the foreign key for a constraint, enter:

```
FOREIGN-KEY
```

followed by clauses that define the columns that comprise the foreign key, that is:

- Specifying that one or more ITEMs and/or GROUPs each define a single column, see "Defining a Label on a Table or Column" on page 429
- Specifying that each ITEM contained in a GROUP defines one column, see "Naming Columns" on page 432.

The columns comprising the foreign key must already be defined in the CONTAINS clause, as the foreign key must exist as a column of the table.

You can name the foreign key column using the KNOWN-AS clause. When you generate an SQL statement the name defined in the KNOWN-AS clause is checked against the column generated. The column name and foreign key name must be the same.

For example, the columns of a table are generated from an expanded GROUP member. However the foreign key comprises only one of the table's columns.

The DB2-TABLE definition includes the clauses:

```
CONTAINS MANY-ITEMS EXPAND
CONSTRAINT FOREIGN-KEY ITEM4 KNOWN-AS COL-4
```

Although the CONTAINS clause does not name the member ITEM4, the GROUP, MANY-ITEMS, includes the clauses:

```
    CONTAINS
    ITEM1 KNOWN-AS COL-l
    , ITEM2 KNOWN-AS COL-2
    , ITEM3 KNOWN-AS COL-3
    , ITEM4 KNOWN-AS COL-4
```

The generated table has four columns, COL_l, COL_2, COL_3, and COL_4. The foreign key column is COL_4.

You can clarify correspondence between members in the repository where:

- Foreign key column names differ from corresponding primary key column names
- Different ITEM and/or GROUP members represent the foreign key columns in one table and the corresponding primary key columns in another table

except where the foreign key is defined using an expanded GROUP. To specify the column(s) in another table to which the foreign key refers, enter:

```
MEMBER member
```

where *member* is the name of an ITEM or GROUP member defining the column(s).

Where the name of a foreign key column(s) is different from its name in another table, you can optionally specify the local name of the column using the KNOWN-AS clause.

To define the table to which a foreign key refers, enter:

```
REFERENCES table
```

where *table* is the name of a DB2-TABLE or SQL-TABLE member.

To successfully generate SQL statements:

- One REFERENCES clause must be defined for each foreign key specified
- The referenced DB2-TABLE must exist and include a valid CREATOR-OWNER clause.

To define the column name for the constraint, enter:

```
COLUMNS column-name
```

where *column-name* is the name of a column in a DB2-TABLE or SQL-TABLE member.

To define the delete rule for the constraint, enter:

```
DELETE option
```

where *option* is RESTRICT, CASCADE, or SET-NULL.

The keywords are similar to those used by DB2 and they have the same meanings.

If you do not define a DELETE rule, the DB2 default applies.

Defining the Encoding Scheme

To specify that the data is to encoded using the ASCII CCSID specified during installation, enter:

```
CCSID ASCII
```

To specify that the data is to be encoded using EBCDIC CCSID specified during installation, enter:

```
CCSID EBCDIC
```

To specify that the data is to be encoded using UNICODE CCSID specified during installation, enter:

```
CCSID UNICODE
```

Defining an Edit Routine

To define an edit routine for the table, enter:

```
EDITPROC process
```

where *process* is the name of a SYSTEM, MMR-SYSTEM, PROGRAM, or MODULE member.

The member represents an edit routine that must exist in DB2. In DB2, the edit routine is invoked whenever a row in the table is retrieved, updated or inserted.

Defining a Validation Routine

To define a validation routine for the table, enter:

```
VALIDPROC process
```

where *process* is the name of a SYSTEM, MMR-SYSTEM, PROGRAM, or MODULE member.

The member represents a validation routine that must exist in DB2. In DB2, the validation routine receives an entire table row as input and may be used to control a subsequent INSERT, UPDATE, or DELETE statement.

Defining an Audit Option

To define the type of DB2 audit that you wish to be carried out on the table, enter:

```
AUDIT option
```

where *option* is NONE, CHANGES, or ALL.

The keywords are the same as in DB2 and have the same meaning.

If you do not define an AUDIT clause the DB2 default applies.

Specifying the Estimated Number of Rows in a Table

If you want to calculate the total size of a table, using the DB2-SIZE command, you have to specify the number of rows that it contains.

To specify the number of rows that the table contains, enter:

```
CARDINALITY integer
```

where *integer* is an integer of no more than 18 digits.

If you do not define a CARDINALITY clause a default of 1 is assumed when you issue a DB2-SIZE command on the DB2-TABLE. The cardinality defined in the member definition, or the default assumed, can be overridden by the CARDINALITY keyword in the DB2-RECALCULATE command.

If the CARDINALITY you define is an estimate of the number of rows, the resulting total size is also an estimate.

Specifying an Object Identifier

You can assign an Object Identifier to be used for the Table by entering:

```
OBID integer
```

where *integer* is a number consisting of up to 18 digits.

The OBID keyword is needed only if ROSHARE READ is selected. If ROSHARE READ is not selected for this Table, the integer must not identify an existing or previously used Object Identifier.

Specifying Data Capture

To specify that the logging of SQL INSERT, UPDATE, and DELETE operations on the Table is augmented by additional information enter:

```
DATA CAPTURE CHANGES
```

To specify that no additional data is added to the log enter: `DATA CAPTURE NONE`.

This is the default setting.

Example of a DB2-TABLE Definition and Generated SQL Statement

```
ADD CUST-TABLE-T;                                  1
DB2-TABLE
 IN DB2-TS1                                        2
 CREATOR-OWNER SLJ1                                1

DB2-COMMENT "CUSTOMERS" NAMES, ADDRESSES"          10
             "AND CODES"

DB2-LABEL 'CUSTOMER TABLE'                         11

 COLUMN HELD-AS
  CONTAINS
      IT-COL01 3 KNOWN-AS CUST-NO                  3
          FOR-BIT-DATA
          PRIMARY KEY                              4
          NOT-NULL
          FIELDPROC PM-FP1-1
          DB2-LABEL 'CUSTOMER-NUMBER'              5

     ,IT-COL02 2                                   3
          WITH-DEFAULT
          PRIMARY-KEY                              4

     ,IT-COL03 1                                   3
          FIELDPROC PM-FP1-2 CONSTANT
            'CONST1, CONST2'

     ,IT-COL04 3                                   4
          PRIMARY-KEY                              5
          NOT-NULL

     ,(5) IT-COL04                                 6

     ,IT-GRP02
          DB2-COMMENT 'CUSTOMER ADDRESS'           7

CONSTRAINT
  NAMED CNSTRNT1
  FOREIGN-KEY                                      8
   IT-COL01 KNOWN-AS CUST-NO
    MEMBER IT-COL04  KNOWN-AS CUSTCODE
    REFERENCES ORDER-TABLE

EDITPROC PR-EP1-1
VALIDPROC PM-VP1-1                                 9
AUDIT NONE


CARDINALITY 600000

;
```

Repository Definition

```
CREATE TABLE SLJ1.CUST_TABLE_T                          1
  (CUST_NO        CHAR(15)                              3
    FOR BIT DATA                  NOT NULL
    FIELDPROC PM_FP1_1


,IT_COL02         DECIMAL(11.5) NOT NULL WITH DEFAULT   3
,IT-COL03         VARCHAR(150)                          3
  FIELDPROC PM_FP1_2 (CONST1, CONST2)
,IT-COL04         FLOAT(21)      NOT NULL               3

,IT-COL04_1       FLOAT(21)
,IT-COL04_2       FLOAT(21)                             6
,IT-COL04_3       FLOAT(21)
,IT-COL04_4       FLOAT(21)
,IT-COL04_5       FLOAT(21)

,IT_GRP02              CHAR(100)

,PRIMARY KEY       (CUST_NO
                            ,IT_COL02                   4
                            ,IT_COL04

,FOREIGN KEY CNSTRNT1
                            (CUST_NO                    8
         REFERENCES SLJ1.ORDER_TABLE)

) IN DB_CUST.TS_CUST01                                  2

EDITPROC PM_EP1_1
VALIDPROC PM_VP1_1                                      9
AUDIT NONE


LABEL ON TABLE SLJ1.CUST_TABLE_T
        IS 'CUSTOMER TABLE'                             10
;

LABEL ON SLJ1.CUST_TABLE_T
        (CUST_NO IS 'CUSTOMER_NUMBER')                  5
;

COMMENT ON TABLE SLJ1.CUST_TABLE_T
        IS 'CUSTOMERS' NAMES, ADDRESSES AND CODES'      11
;

COMMENT ON TABLE SLJ1.CUST_TABLE-T
        (IT_GRP02 IS 'CUSTOMER ADDRESS')                12
;
```

Generated SQL Statement

1 The derived name for the table is the DB2-TABLE member name qualified by the owner's DB2 authorization ID.

2 The name of the table space to which the table belongs is taken directly from the repository definition and qualified by the database name.

3 A column-name is taken from the KNOWN-AS name of the ITEM specified in the CONTAINS clause, or from the name of the ITEM itself, if no KNOWN- AS name is specified.

4 All the ITEMs that are specified as being part of the primary key are brought together in the SQL statement.

5 The label on the column is taken directly from the DB2-LABEL clause in the member definition.

6 The five occurrences of IT-COL04 specified in the repository definition result in the generation of five columns, each called IT-COL04; each column is uniquely identified by the integer suffixed to it.

7 The comment on the column is taken directly from the DB2-COMMENT clause in the member definition.

8 Details of the foreign key are taken directly from the CONSTRAINT clause of the member definition.

9 These clauses are generated directly from the corresponding member definition.

10 The DB2-COMMENT associated with the table is taken directly from the specification in the member definition.

11 The DB2-LABEL associated with the table is taken directly from the specification in the member definition.

Note:

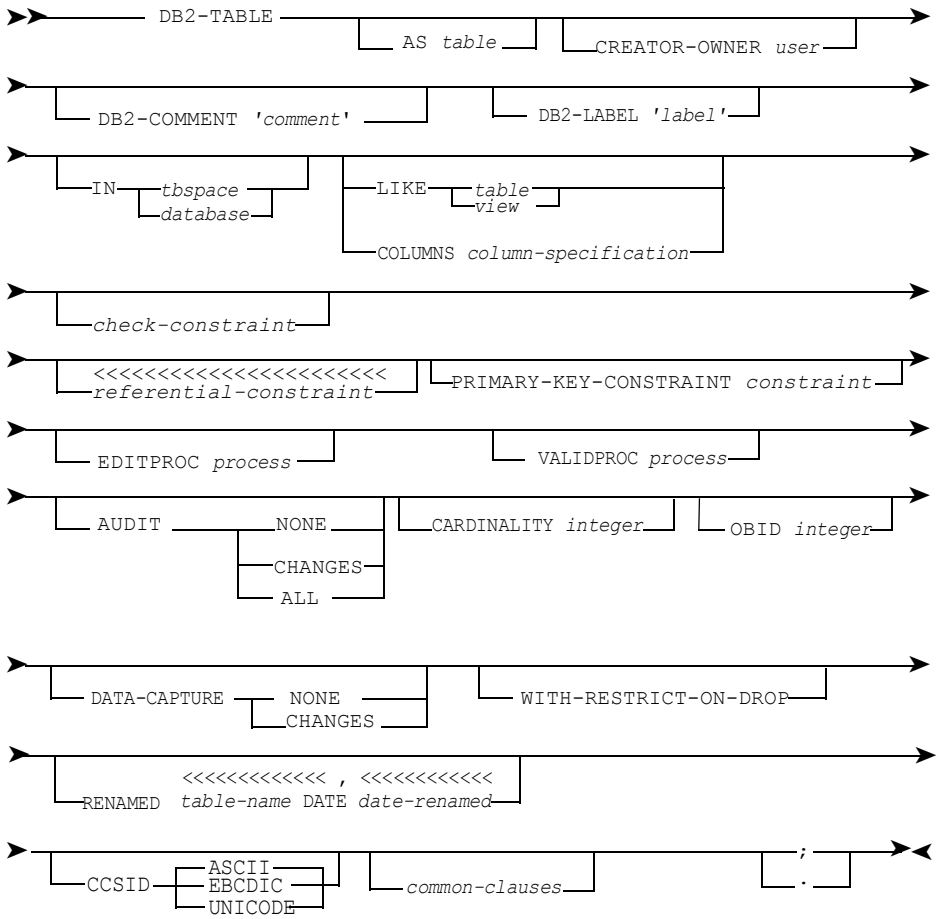
The automatic generation of SQL COMMENT statements depends upon:

- Specifying the WITH-COMMENT keyword in the DB2 command
- The systems administrator having tailored the command variable CM_COMMENTOPT in your DB2 profile,

and the automatic generation of SQL LABEL statements depends upon:

- Specifying the WITH-LABEL keyword in the DB2 command
- The systems administrator having tailored the command variable CM_LABELOPT in your DB2 profile.

DB2-TABLE Syntax



where:

`AS` *`table`* is the name of a DB2-TABLE member.

`CREATOR-OWNER` *`user`* is the name of a DB2-USER member to be used as the authentication ID of this table.

`DB2-COMMENT '`*`comment`*`'` is the comment for this table. This comment is used to generate the SQL COMMENT ON statement.

`DB2-LABEL '`*`label`*`'` is the label for this table. This label is used to generate the SQL LABEL ON statement.

`IN` *`tbspace`* is the name of a DB2-TBSPACE member.

`IN` *`database`* is the name of a DB2-DATABASE member.

`LIKE` *`table`* is the name of a DB2-TABLE member.

`LIKE view` is the name of a DB2-VIEW member.

`PRIMARY-KEY-CONSTRAINT constraint` is the constraint name of the primary key, and must not exceed 128 characters.

`EDITPROC process names` a SYSTEM, PROGRAM, or MODULE member that is the edit routine for the table.

`VALIDPROC process names` a SYSTEM, PROGRAM, or MODULE member that is the validation exit routine for the table.

`CARDINALITY integer` is an integer of no more than 18 digits that specifies the number of rows that the table contains.

`OBID integer` is an integer up to 18 digits to be used as the object identifier for the table.

`RENAMED table-name` is the original member or alias name of this table that was renamed, and `date-renamed` is the date when the rename was executed. This clause is normally inserted by executing the AMEND command that is generated as output from the DB2 RENAME export command.

`table` is the name of a DB2-TABLE member.

`user` is the name of a DB2-USER member.

`comment` is a comment of no more than 254 characters.

`label` is a label of no more than 30 characters.

`tbspace` is the name of a DB2-TBSPACE member.

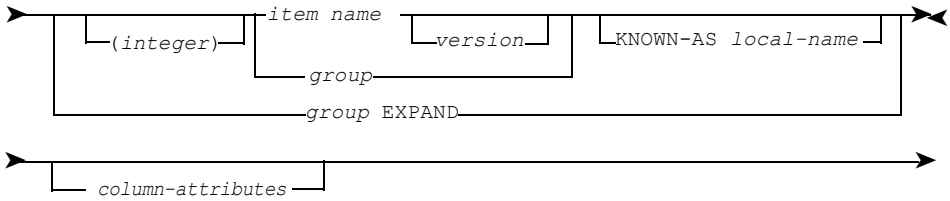
`table` is as defined above.

`view` is the name of a DB2-VIEW member.

`column-specification` is:

```
                                                  <<<< <<, <<<<<<<
 ►──┬──────────────
```

contains-clause is:



where:

(*integer*) is the number of columns to be derived from this item.

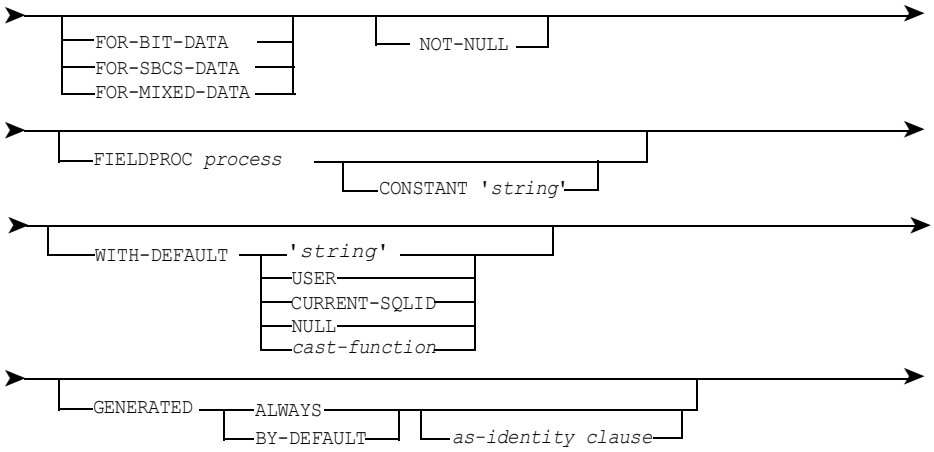
item-name is the name of an ITEM member.

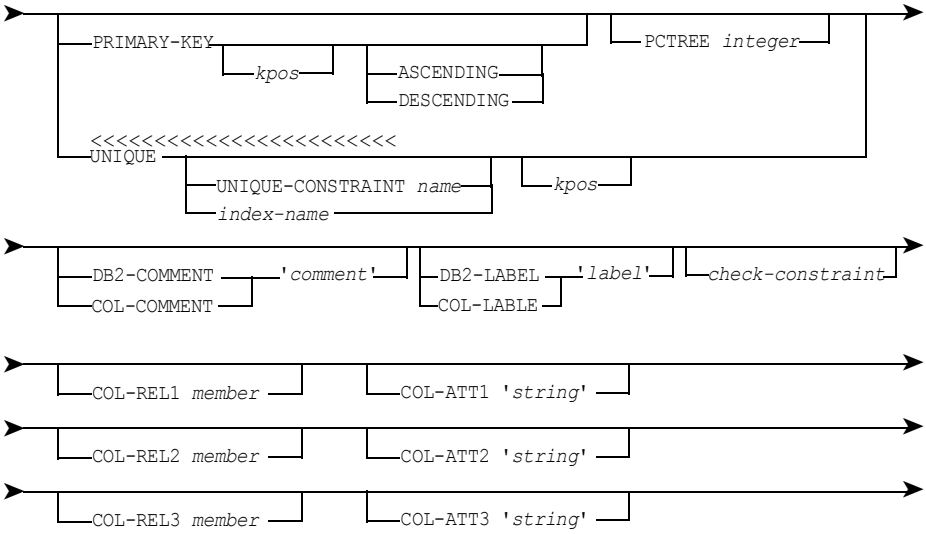
version is an integer between 1 and 15 that is the version number of the form description for this item.

group is the name of a GROUP member.

local-name is the name to be given this column that is different from the name of the item. If multiple columns are derived from this item, the name is suffixed by its sequence number.

column-attributes is:





where:

`FIELDPROC` *`process names`* a SYSTEM, PROGRAM, or MODULE member that is the field procedure exit routine for the table.

`CONSTANT '`*`string`*`'` is the parameter that is passed to the field procedure when it is invoked.

`WITH-DEFAULT '`*`string`*`'` specifies a constant as the default value for the column, and must match the datatype of the column.

`kpos` is the key column order number to be used if it is different from the order in which the columns are defined in the table.

`PCTFREE` *`integer`* is an integer from 0 to 99 and defines the percentage of free space in the primary key.

`UNIQUE-CONSTRAINT` *`name`* is the constraint name for this unique key with up to 128 characters.

`unique-index` is the index name for this unique key.

`kpos` is as described above.

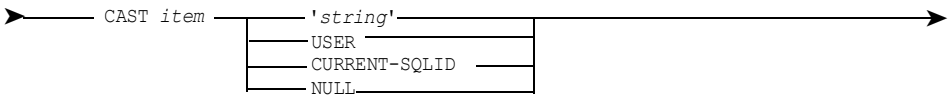
`DB2-COMMENT '`*`comment`*`'` is the comment for this column. This comment is used to generate the SQL COMMENT ON statement for this column.

`DB2-LABEL '`*`label`*`'` is the label for this column. This label is used to generate the SQL LABEL ON statement for this column.

`COL-RELn member` is the name of a repository member that has a relationship with this column of this relational table.

`COL-ATTn 'string'` is a string of up to 254 characters defining additional attributes for this column.

cast-function is:

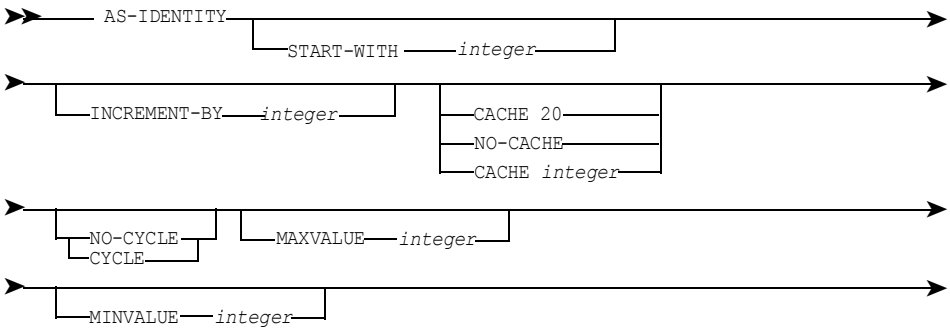


where:

item is the name of an ITEM that is defined as a DISTINCT-TYPE.

'string' is the default value to be used of up to 254 characters.

as-identity clause:



where:

`START-WITH integer` is a numeric integer value of up to 31 digits, positive or negative, and must be within MINVALUE and MAXVALUE.

`INCREMENT-BY integer` is a numeric integer value of up to 31 digits, positive or negative.

`CACHE integer` is an integer value of no more than 18 digits.

`MAXVALUE integer` is a numeric integer value of up to 31 digits, positive or negative, and must be larger than MINVALUE.

`MINVALUE integer` is a numeric integer value of up to 31 digits, positive or negative, and must be smaller than MAXVALUE.

check-constraint:

```
>>──┬──────────────────────────────────────────────────────────────┬──><
   └─CHECK-CONSTRAINT NAMED constraint ──┬─<<
```

```
>>---MEMBER---+-item-+---+-----------------------------+---><
              +-group-+   +-KNOWN-AS local-name-------+
```

item is the name of an ITEM member.

local-name is a name of up to 18 characters.

group is the name of a GROUP member.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-TBSPACE

DB2-TBSPACE defines DB2 table spaces in the repository.

Refer to "DB2-TBSPACE Syntax" on page 467 for the syntax of the DB2-TBSPACE member definition.

To define a DB2 table space:

```
DB2-TBSPACE
```

The member definition must begin with this member type identifier. All other clauses available to define DB2-TBSPACE members are optional. However, the IN clause defining the database to which the table space belongs, must be present for the successful generation of SQL CREATE TABLESPACE, DROP TABLESPACE, or ALTER TABLESPACE statements.

In DB2, if a table space is partitioned, the corresponding index must also be partitioned. Therefore for DB2-TBSPACE members defined with a PARTITION clause, you should define a corresponding clustered DB2-INDEX with the same partitions.

Note:

A DB2-TBSPACE member definition can contain either the PARTITION clause or the SEGSIZE clause. It cannot contain both, as a table space may not be partitioned and segmented at the same time.

Refer to Chapter 4, "Repository Definition," on page 91 for details of generating table space names.

Defining a Large Table Space

To define a table space as that can hold more than 64 GB of data, enter:

`LARGE` or `LOB`

This partitioned table space can hold either compressed or uncompressed data with a maximum of 254 partitions and a maximum size of 4 GB per partition. If LARGE is specified, NUMPARTS must be specified. If LARGE is omitted and the value for NUMPARTS is greater than 64, or LARGE is specified and the value for NUMPARTS is less than 65, then the table space will have the attributes of a large table space.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-TBSPACE member definitions using an AS clause.

Specifying the Database to Which a Table Space Belongs

To specify the database to which a table space belongs enter:

`IN` *`database`*

where *`database`* is the name of a DB2-DATABASE member.

On encoding, the member specified in the IN clause is checked to ensure that it is a DB2-DATABASE member.

The IN clause must be defined to successfully generate SQL CREATE TABLESPACE and CREATE TABLE statements. The DB2-DATABASE member named in the IN clause is used to generate a database-qualified name for the table space.

Defining Storage Space

To define the physical space occupied by a table space or partition you can either:

- Define the VSAM catalog it is to use
- Define the storage group it belongs to

To define the VSAM catalog the table space is to use, enter:

`VCAT` *`catalog`*

where *`catalog`* is the name of a VSAM catalog, of no more than eight characters.

To define the storage group to which the table space or partition belongs, enter:

`STOGROUP` *`stogroup-name`*

where *`stogroup-name`* is the name of a DB2-STOGROUP member.

On encoding, the member specified in the STOGROUP clause is checked to ensure that it is a DB2-STOGROUP member.

The storage group defined in the DB2-DATABASE definition is the default used by table spaces:

- That belong to the database
- That do not have a storage group specified in their own member definition.

When you define the STOGROUP clause, you can additionally define further details of primary and secondary storage using the PRIQTY, SECQTY sub-clauses. Both sub-clauses are expressed in kilobytes.

To specify the amount of primary storage space, enter:

```
PRIQTY p
```

where *p* is the number of kilobytes, and must be in the range 3 to 4194304 inclusive.

To specify the amount of secondary storage space, enter:

```
SECQTY s
```

where *s* is the number of kilobytes, and must be in the range 0 to 131068 inclusive.

Primary and secondary storage space is allocated in the storage area defined either:

- In the VOLUMES clause of the DB2-STOGROUP member named in the STOGROUP clause of the DB2-TBSPACE member
- In the storage group defined in the STOGROUP clause of the DB2- DATABASE

You can also specify whether or not DB2-defined datasets are to be erased when the table space is deleted in a DROP command. If you want the datasets to be erased, enter:

```
ERASE YES
```

If you do not want the datasets to be erased, enter:

```
ERASE NO
```

If you do not specify PRIQTY, SECQTY, or ERASE subclauses, the DB2 defaults apply.

Defining Free Space

You can accommodate future expansion of a table space or partition by defining:

- The frequency with which pages are left free
- The percentage of each page that is left free

To define the relative frequency with which free pages are allocated, enter:

```
FREEPAGE fn
```

where *fn* is an integer in the range 0 to 255.

For example, FREEPAGE 4 means that 1 free page is left after every 4 pages.

To define the percentage space kept free on a page, when a table space or partition is loaded or reorganized, enter:

```
PCTFREE pn
```

where *pn* is an integer in the range 0 to 99.

If you do not define FREEPAGE or PCTFREE clauses, the DB2 defaults apply.

Specifying Member Cluster

To specify that data inserted by the INSERT statement is not clustered by the implicit clustering index (the first index) or the explicit clustering index, enter:

```
MEMBER-CLUSTER
```

When this attribute is used, DB2 chooses where to locate the data in the table space based on available space.

MEMBER-CLUSTER should not be specified for a LOB table space, or a table space in a work file database, or a TEMP database.

Defining Table Space Partitions

There are two ways to define that a table space is to have partitions. One is by coding the NUMPARTS attribute, and the other is to directly code the PARTITION NUMBER attribute.

To define a table space with a specific number of partitions, enter:

```
NUMPARTS integer
```

where *integer* is the total number of partitions in the table space, and can range from 1 to 254 inclusive.

In DB2, NUMPARTS must be specified if DSSIZE is specified and LOB is omitted, or LARGE is specified. The maximum size of each partition depends on the value specified for DSSIZE or LARGE. If DSSIZE or LARGE is not specified, the number of partitions specified determines the maximum size of each partition.

To define the individual partitions, enter:

```
PARTITION NUMBER n partition-clause
```

where *n* is the partition number of the particular partition to which this clause applies, and *partition-clause* is the attributes defined to this clause. If no NUMPARTS is defined but PARTITION NUMBER is, a NUMPARTS is generated automatically in the SQL CREATE TABLESPACE statement generated by the DB2 CREATE command, based on the highest PARTITION NUMBER entered. If neither NUMPARTS nor PARTITION NUMBER is entered, the tablespace is not partitioned.

Specifying User-defined or DB2 Defined Datasets (Using-Block)

The using-block clause consists of the VCAT attribute, or STOGROUP, PRIQTY, SECQTY, and ERASE attributes.

The using-block indicates whether the dataset for the table space is defined by the user or by DB2. If DB2 is to define the dataset, the clause also gives space allocation parameters and an erase rule. If the using-block is omitted, DB2 defines the datasets using the default storage group of the database and the defaults for PRIQTY, SECQTY, and ERASE.

If the table space is partitioned, there can be a using-block for each partition; either one you give explicitly or one provided by default. Generally, the meaning of the clause and the rules that apply to it are the same as for a nonpartitioned table space.

If the datasets for the table space are managed by the user, enter:

```
 VCAT catalog-name
```

where *catalog-name* is a short identifier of up to 8 characters for an integrated catalog facility catalog.

To specify that DB2 will define and manage the datasets for the table space, enter:

```
STOGROUP stogroup
```

where *stogroup* is the name of a DB2-STOGROUP member.

To specify values for primary allocations for datasets, enter:

```
PRIQTY p
```

where *p* is an integer that specifies the minimum primary space allocation for a DB2-managed dataset. The primary space allocation is at least *p* kilobytes.

To specify the minimum secondary space allocation, enter:

```
SECQTY s
```

where *s* is an integer that specifies the minimum secondary space allocation for a DB2-managed dataset. The secondary space allocation is at least s kilobytes.

To indicate whether the DB2-managed datasets for the table space or partition are to be erased when they are deleted when the table space is dropped, to not erase the datasets enter:

```
ERASE YES
```

Or

```
ERASE NO
```

Specifying that DB2 Tracks Modified Pages

To specify that DB2 tracks modified pages in the space map pages of the table space or partition, enter:

```
TRACKMOD YES
```

DB2 tracks changed pages to improve the performance of incremental image copy. YES is the DB2 default unless the table space is in a TEMP database.

Do not specify TRACKMOD for a LOB table space. For a table space in a TEMP database, DB2 uses TRACKMOD NO regardless of the value specified. Enter:

```
TRACKMOD NO
```

if DB2 does not track changed pages in the space map pages.

The TRACKMOD attribute can be coded for the entire table space, or as an attribute of the partition-clause for individual partitions.

Defining an Associated Buffer Pool

To define the buffer pool that table spaces use, enter:

```
BUFFERPOOL bufferpool-name
```

where *bufferpool-name* is one of: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BPl0, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP2l, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

If you do not define a buffer pool in a DB2-TBSPACE, the default defined in the DB2-DATABASE to which the table space belongs, applies.

Defining the Encoding Scheme

To specify that the data is to encoded using the ASCII CCSID specified during installation, enter:

```
CCSID ASCII
```

To specify that the data is to be encoded using EBCDIC CCSID specified during installation, enter:

```
CCSID EBCDIC
```

To specify that the data is to be encoded using UNICODE CCSID specified during installation, enter:

```
CCSID UNICODE
```

Defining Whether Selective Partition Locking is Used

To define selective partition locking (SPL) in a partitioned table space, enter:

```
LOCKPART YES
```

If you specify LOCKPART YES and all conditions required for SPL are met, only the partitions accessed lock. If all conditions for SPL are not met, every partition of the table space locks.

To specify no selecting partition locking (SPL), enter: `LOCKPART NO`

The table space is locked with a single lock on the last partition.

Defining the Maximum Number of Rows on a Data Page

To specify the maximum number of rows on a data page, enter:

```
MAXROWS n
```

where *n* is an integer in the range 1 to 255.

The default is 255. The MAXROWS value for a table space in a work file database cannot be specified.

Defining that Datasets are Left Open or Closed After Use

To define that the dataset, on which a table space resides, is to be closed when not in use, enter:

```
CLOSE YES
```

To define that it is to remain open, enter:

```
CLOSE NO
```

If you do not define a CLOSE clause the DB2 default applies.

Defining a Dataset Password

To define a password for the VSAM dataset, on which a table space resides, enter:

```
DSETPASS password
```

where *password* is a VSAM dataset password of no more than eight characters.

Defining the Size of a Locking Storage Unit

To define the size of the storage unit that can be locked, enter:

```
LOCKSIZE unit
```

where *unit* is one of these units of locking:

```
ANY
PAGE
TABLESPACE or S
TABLE
ROW
LOB
```

If you do not specify a LOCKSIZE the DB2 default applies.

Defining the Maximum Number of Page or Row Locks

To define the maximum number of page or row locks that an application process can hold simultaneously in the tablespace, enter:

```
LOCKMAX unit
```

where *unit* is one of the following: 0, integer or SYSTEM. For the meaning of these values refer to the IBM documentation.

Note:

LOCKMAX can be defined only if PAGE or ROW is specified in LOCKSIZE.

Defining Table Space Segments

To define a segmented table space and the number of pages to be allocated to each segment, enter:

```
SEGSIZE integer
```

where *integer* is a multiple of 4, in the range 4 to 64 inclusive.

Note:

If you define a SEGSIZE clause you cannot define a PARTITION clause, as a segmented table space cannot be partitioned.

Defining Whether Tablespace is to be Compressed

To define that the rows in a tablespace or tablespace partition are to be compressed enter:

```
COMPRESS YES
```

To define no data compression for the tablespace or partition enter:

```
COMPRESS NO
```

Defining Group Buffer Pool Usage

To define that all pages are to be cached in the group buffer pool, enter:

```
GBPCACHE ALL
```

To define that only updated pages are to be cached in the group buffer pool, enter:

```
GBPCACHE CHANGED
```

To add no pages into the group buffer pool, enter:

```
GBPCACHE NONE
```

To store only changed system pages enter:

```
GBPCACHE SYSTEM
```

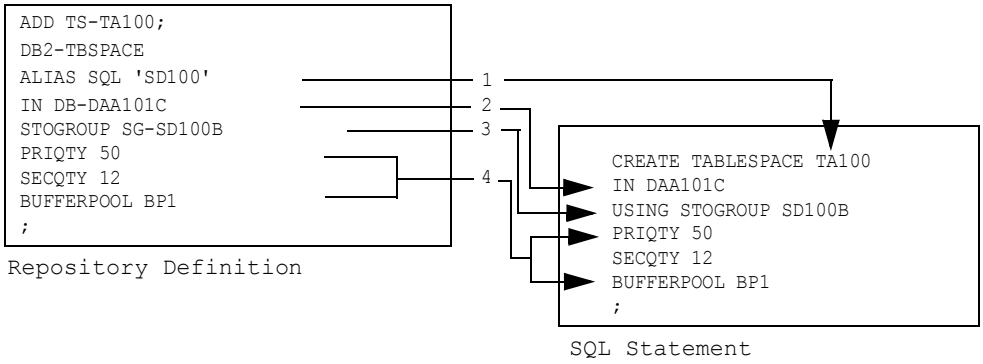
To define logging of space map pages, enter: `TRACKMOD YES` to monitor changes or `TRACKMOD NO` to ignore changes.

To define logging of LOB changes, enter: `LOG YES` to activate logging for LOB columns up to 1GB or `LOG NO` to not log changes.

To define the maximum capacity of a partition valid values are: 1GB, 2GB, 4GB, 8GB, 16GB, 32GB, and 64GB.

Examples

The DB2-TBSPACE member TS-TAl00 is defined in the repository using an ADD command. It represents an unpartitioned table space. An SQL statement is generated from the member definition using a DB2 CREATE command.



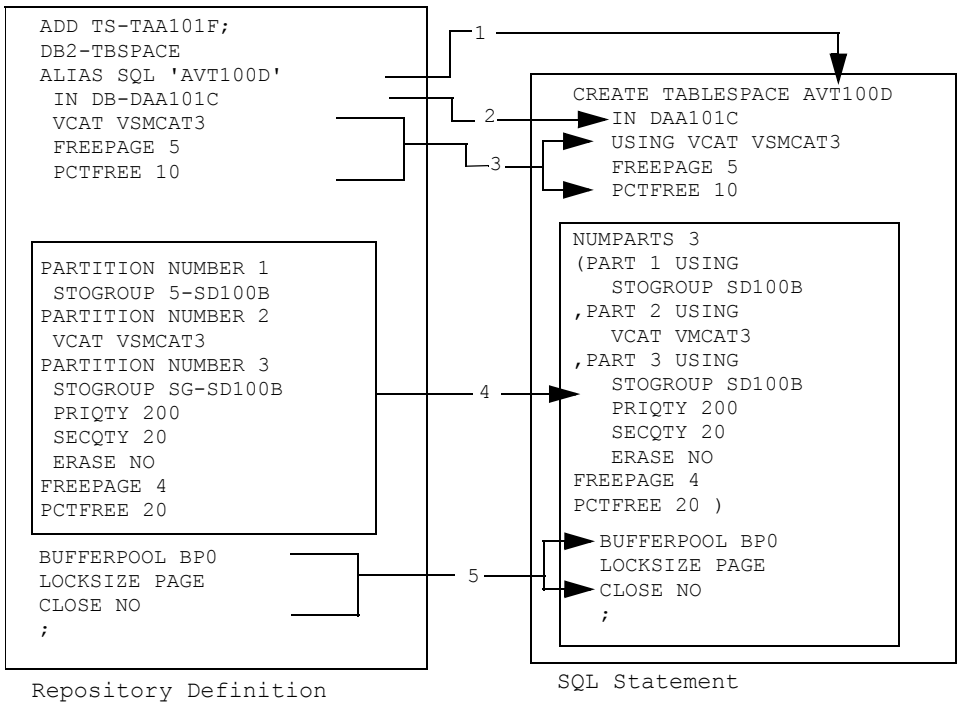
1 The table space name TA 100 is taken from the SQL ALIAS.

2 The database name DAA101C is taken from the SQL ALIAS defined in the DB2-DATABASE member DB-DAA101C.

3 The storage group name SD100 is taken from the SQL ALIAS defined in the DB2-STOGROUP member SG-SDl00B.

4 The PRIQTY, SECQTY and BUFFERPOOL parameters are taken from the repository definition.

The DB2-TBSPACE member TS-TAAl01F is defined in the repository using an ADD command. The member represents a partitioned table space. An SQL statement is generated from the member definition using a DB2 CREATE command.



1 The table space name AVT100D is taken from the SQL ALIAS.

2 The database name DAA101C is taken from the SQL ALIAS defined in the DB2-DATABASE member DB-DAA101C.

3 The VCAT catalog name and the free space parameters are taken from the repository definition.

4 The partition specifications are taken directly from the repository definition, (except for the storage group names, which are taken from the SQL ALIAS defined in the relevant DB2-STOGROUP member definitions.) The NUMPARTS 3 clause is generated automatically.

5 The BUFFERPOOL, LOCKSIZE and CLOSE parameters are taken from the repository definition.

DB2-TBSPACE Syntax

```
>>-DB2-TBSPACE-+-LARGE-+-+-AS tbspace-+-+-IN dbname-+
               +-LOB---+

>-+-using-block-+-+-free-block-+-+-gbpcache-block-+

>-+-MEMBER-CLUSTER-+-+-NUMPARTS integer-+-+-<<<<<<<<<<<<<<-+
                    +                   +  +-partion-clause-+

>-+-SEGSIZE integer-+-+-BUFFERPOOL bpname-+-+-LOCKSIZE-+-ANY-------+
                      +                   +           +-TABLE-----+
                                                      +-TABLESPACE+
                                                      +-S-------+
                                                      +-ROW-----+
                                                      +-PAGE----+
                                                      +-LOB-----+

>-+-LOCKMAX-+-0-------+-+-LOCKPART-+-YES-+-+-MAXROWS MR-+
            +-integer-+ +           +-NO--+
            +-SYSTEM--+

>-+-LOG-+-YES-+-+-TRACKMOD-+-YES-+-+-COMPRESS-+-YES-+-+
        +-NO--+ +          +-NO--+           +-NO--+

>-+-DSETPASS password-+-+-DSSIZE integer-+-+-CCSID-+-ASCII-+-+
                      +                  +         +-EBCDIC+
                                                   +-UNICODE+

>-+-CLOSE-+-YES-+-+-common-clauses-+-+-;-+-><
          +-NO--+                   +-.-+
```

where:

tbspace is the name of a DB2-TBSPACE member

dbname is the name of a DB2-DATABASE member.

using-block is:

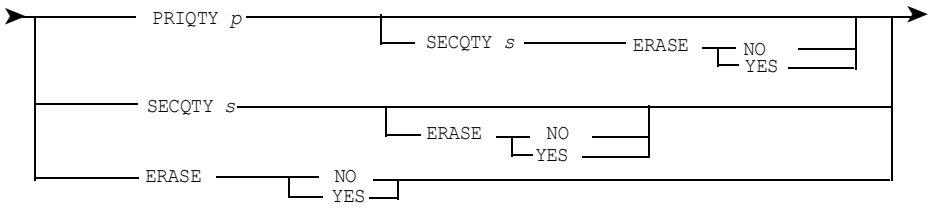
```
>-+-VCAT catalog-----------------------------------+
  +-STOGROUP stogroup-+-PRIQTY p-+-+-SECQTY s-+-+-ERASE-+-NO-+-+
                      +          +           +         +-YES-+
```

where:

catalog-name is a VSAM catalog of no more than 8 characters.

stogroup-name is the name of a DB2-STOGROUP member.

space-specification is:

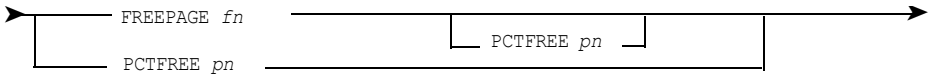


where:

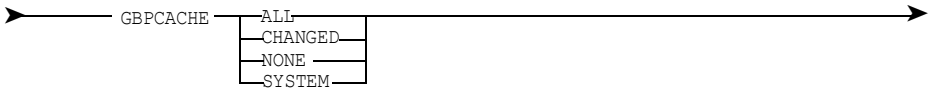
p is an integer in the range 3 to 4194304.

s is an integer in the range 0 to 4194304.

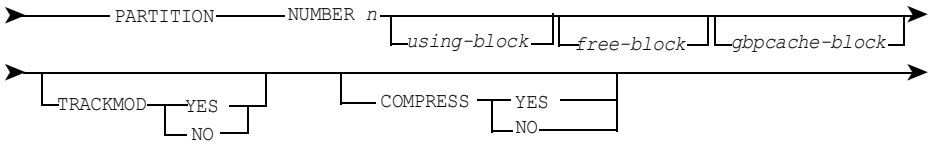
free-block:



gbpcache-block:



partition-clause:



where:

`NUMPARTS` *`integer`* is a positive integer that specifies the total number of partitions in the table space.

`PARTITION NUMBER` *`n`* is a positive integer identifying a particular partition. If NUMPARTS was entered, n must not exceed the NUMPARTS value.

The using-block, free-block, gbpcache-block, and the trackmod block can be specified at a higher level for non-partitioned tablespaces, or can be specified as subclauses of the partition clause for partitioned tablespaces.

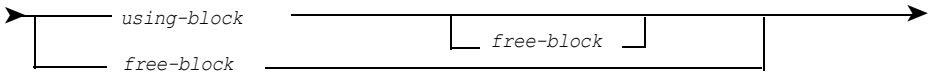
where:

fn is an integer in the range 0 to 255

pn is an integer in the range 0 to 99.

n is an integer in the range 1 to 64

storage is:



where:

using-block is defined above

free-block is defined above

bpname is one of: BP0, BP1, BP2, BP3, BP4, BP5, BP6, BP7, BP8, BP9, BPl0, BP11, BP12, BP13, BP14, BP15, BP16, BP17, BP18, BP19, BP20, BP21, BP22, BP23, BP24, BP25, BP26, BP27, BP28, BP29, BP30, BP31, BP32, BP33, BP34, BP35, BP36, BP37, BP38, BP39, BP40, BP41, BP42, BP43, BP44, BP45, BP46, BP47, BP48, BP49, BP32K, BP32K1, BP32K2, BP32K3, BP32K4, BP32K5, BP32K6, BP32K7, BP32K8, BP32K9, BP8K0, BP8K1, BP8K2, BP8K3, BP8K4, BP8K5, BP8K6, BP8K7, BP8K8, BP8K9, BP16K0, BP16K1, BP16K2, BP16K3, BP16K4, BP16K5, BP16K6, BP16K7, BP16K8, BP16K9.

password is a VSAM dataset password, of no more than 8 characters

mr is an integer in the range 1 to 255

integer is an integer, which must be a multiple of 4, in the range 4 to 64.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of *common clauses*.

DB2-TEMPTABLE

DB2-TEMPTABLE defines DB2 temporary tables in the repository. Temporary tables are a special subset of the DB2 table, and defines a temporary table for the current application process.

See "DB2-TEMPTABLE Syntax" on page 472 for the DB2-TEMPTABLE member definition syntax.

Defining a Temporary Table

To define a DB2 temporary table, enter:

```
DB2-TEMPTABLE TEMPORARY
```

followed by the attributes for this table.

Defining an Owner

To define an owner of this table, enter:

```
CREATOR-OWNER user
```

where *user* is a DB2-USER member that identifies the user with the authentication ID for this table. This attribute is required in order to generate an SQL statement using an export command such as DB2 CREATE and DB2 ALTER.

Defining a Comment or Label

To define a DB2 comment, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is a string no longer than 254 characters in length.

To define a DB2 label, enter:

```
DB2-LABEL 'label'
```

where *label* is a string no longer than 254 characters in length.

Specifying a DB2 Table Which Describes the Temporary Table

To specify a table or view whose columns have the same name, data type, and attributes as the columns of the temporary table, enter:

```
LIKE name
```

where *name* is the name of a DB2-TABLE or DB2-VIEW member. If a table is entered, the column default attributes are also defined by that table. The identified table must *not* be an auxiliary table or a temporary table.

Specifying the Columns Without Using LIKE

If the LIKE attribute is not used and the columns of this temporary table are to be defined individually, enter:

```
COLUMNS form-keyword  CONTAINS item1, item2, ...
```

where *form-keyword* is one of these:

- ENTERED-AS
- HELD-AS
- REPORTED-AS
- DEFAULTED-AS

To specify the ITEM member that defines this column, enter:

```
CONTAINS item
```

where *item* the name of an ITEM member which identifies the column. Each item can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

where *version* is an integer in the range of 1 to 15, and defines the form description version.

To define multiple columns with identical attributes using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) item
```

where *integer* is the number of columns to be derived from this member.

To define multiple columns from items contained within a group, enter:

```
CONTAINS group EXPAND
```

where *group* is the name of a GROUP member that contains the items desired. All items defined in the group are included into the table.

You can explicitly name the column if you do not want its name to be derived from the ITEM name or its alias. To define the name of the column, enter:

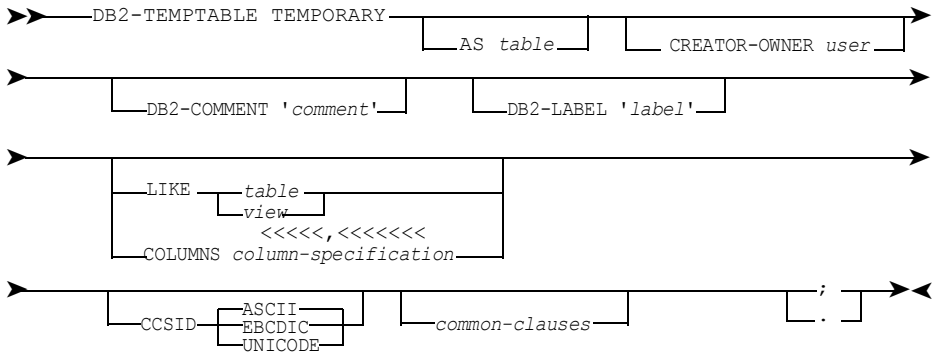
```
KNOWN-AS name
```

where *name* is a string of no more than 18 characters.

To specify that the column contains no null values, enter:

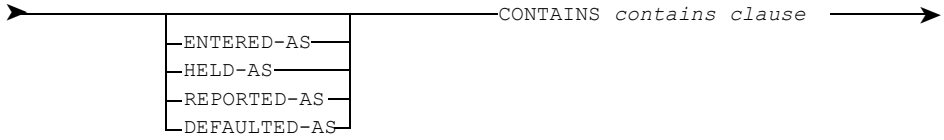
```
CONTAINS item NOT-NULL
```

DB2-TEMPTABLE Syntax

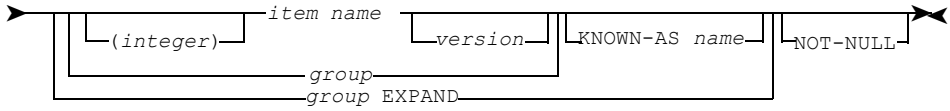


where:

column-specification:



contains-clause:



DB2-TRIGGER

DB2-TRIGGER defines DB2 triggers. Create triggers in a schema and build trigger packages at the current server in a DB2 environment.

Refer to "Reusing Existing Member Definitions" on page 509 for details on reusing all or part of existing DB2-TRIGGER member definitions using an AS clause.

A DB2 trigger defines a set of actions that are executed when a delete, insert, or update operation occurs on a specified table.

Triggers can be used along with referential constraints and check constraints to enforce data integrity rules. Triggers are more powerful than constraints because they can also be used to cause updates to other tables, automatically generate or transform values for inserted or updated rows, or invoke functions that perform operations both inside and outside of DB2.

Defining a Trigger

To define a DB2-TRIGGER, enter:

```
DB2-TRIGGER
```

followed by the attributes for this trigger.

Defining an Owner

To define the owner of this trigger, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the trigger. This clause is required for the successful generation of SQL statements, but can be overridden by an SQLID clause in the DB2 export command.

Defining a Comment

To define a comment for this trigger, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is the comment for this trigger. This value is used to generate the SQL COMMENT ON statement when executing an export command.

Specifying When the Trigger is Executed

To specify that the trigger is a before trigger, enter:

```
OPTION BEFORE
```

DB2 executes the triggered action before it applies any changes caused by an insert, delete, or update operation on the subject table.

To specify that the trigger is an after trigger, enter:

```
OPTION AFTER
```

DB2 executes the triggered action after it applies any changes caused by an insert, delete, or update operation on the subject table.

Specifying What Action Executes the Trigger

To specify that the trigger is activated on an insert, enter:

```
ACTION INSERT ON table
```

where *table* is the name of a DB2-TABLE member which is the subject table. DB2 executes the triggered action whenever there is an insert operation on the subject table.

To specify that the trigger is a delete trigger, enter:

```
ACTION DELETE ON table
```

DB2 executes the triggered action whenever there is a delete operation on the subject table.

To specify that the trigger is an update trigger, enter:

```
ACTION UPDATE ON table
```

DB2 executes the triggered action whenever there is an update operation on the subject table.

Specifying the Old or New Reference Table

The triggered action may refer to the values in the affected table rows. This action is supported through the use of transition variables and transition tables. Transition variables use the names of the columns in the subject table qualified by a specified name that identifies whether the reference is to the old value (before the update) or the new value (after the update).

To specify the correlation names for the table names for the transition tables or the name of a temporary table that identifies the rows to be modified by the trigger, enter:

```
REF-OLD 'string'
```

where '*string*' is up to 254 characters and specifies the correlation name or the name of a temporary table that identifies the state of the rows prior to the triggering SQL operation.

```
REF-NEW 'string'
```

where '*string*' is up to 254 characters and specifies the correlation name or the name of a temporary table that identifies the state of the rows as modified by the triggering SQL operation.

Specifying That the Trigger Applies to Rows

To specify that DB2 executes the triggered action for each row of the subject table that the triggering SQL operation modifies, enter:

```
ROW-TRIGGER YES
```

If the triggering SQL operation does not modify any rows, the triggered action is not executed.

To specify that DB2 executes the triggered action only once for the triggering SQL operation even if the triggering SQL operation does not modify any rows, enter:

```
ROW-TRIGGER NO
```

Do *not* specify ROW-TRIGGER NO for a before trigger.

Specifying the Trigger Condition

Triggers can be executed based on a search condition. The triggered SQL statements are executed only if the search condition evaluates to true, or if the condition is omitted.

To specify a condition, enter:

```
CONDITION 'condition'
```

where *condition* is the search condition of up to 254 characters.

Specifying the Triggered-Action (SQL statements)

To specify the action to be performed when the trigger is activated, enter:

```
SQL 'text'
```

where *text* is the SQL statements of up to 254 characters that comprise the triggered action.

DB2-TRIGGER Syntax

```
>>-----DB2-TRIGGER--+-------------+--+-------------------------+-->
                    |-AS alias----|  |-CREATOR-OWNER user------|

>--+-------------------------+--+-------------------------+-->
   |-DB2-COMMENT 'comment'---|  |-OPTION--+-BEFORE--+-----|
                                          |-AFTER-----|

>--ACTION--+-INSERT--+--+-ON table--+--+-REF-OLD--+-'string'-+--+
           |-DELETE--|  |           |  |-REF-NEW--|           |
           |-UPDATE--|  |           |  |          |           |

>--+-ROW-TRIGGER--+-YES--+--+-CONDITION 'condition'-+--+-SQL 'text'-+--+--;--+--><
   |                |-NO--|  |                      |  |            |  |--.--|
```

DB2-COMMENT '*comment*' is a comment of no more than 254 characters. This comment is used to generate the SQL COMMENT ON command for this function.

where:

alias is the name of a DB2-TRIGGER member.

user is the name of a DB2-USER member.

table is the name of a DB2-TABLE member that is the triggering table.

string is a maximum of 254 byte string specifying the referencing clause.

text is the trigger SQL action sequence.

DB2-USER

DB2-USER defines DB2 authorization IDs in the repository.

Refer to "DB2-USER Syntax" on page 480 for the syntax of DB2-USER member definition.

In the DB2 environment all objects have an owner, who may also be the creator of the object. DB2-USER members document the owner or creator of DB2 objects.

To define an authorization ID, enter:

```
DB2-USER
```

The member definition must begin with this member type identifier.

DB2-USER members are specified as the recipients of privileges in DB2-PRIVILEGE members. DB2-USER members are also used to generate qualified names for DB2-ALIAS, DB2-INDEX, DB2-TABLE, and DB2-VIEW members that have a CREATOR-OWNER clause defined.

Note:

The SQLID keyword in DB2 commands allows you to override the user name specified in the CREATOR-OWNER clause.

A DB2-USER member can be defined to represent a group of users, for example a project team. The DB2-USER member representing the group ID has a CONTAINS clause listing the DB2-USER members representing each member of the project team.

You can generate SQL CREATE SYNONYM statements from DB2-USER members if you define synonyms in the SYNONYMS clause.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part of existing DB2-USER or SQL-USER member definitions using an AS clause.

Defining a Location

To define the location to which a user belongs, enter:

```
LOCATION location-name
```

where *location-name* is the name of a DB2-LOCATION member that represents a local or remote location in a distributed network.

In DB2-USER member definitions, the LOCATION clause is used to generate location-qualified names for tables and views belonging to a particular user. The location name defined in a DB2-USER member can be overridden by a LOCATION clause defined in a DB2 command.

Defining a Group

To define a group ID, enter:

```
CONTAINS user-list
```

where *user-list* is the name of one or more DB2-USER or SQL-USER members, separated by commas.

Each DB2-USER member represents one authorization ID, and together represent all the users who can use the group ID.

DB2-USER members containing group IDs can be nested.

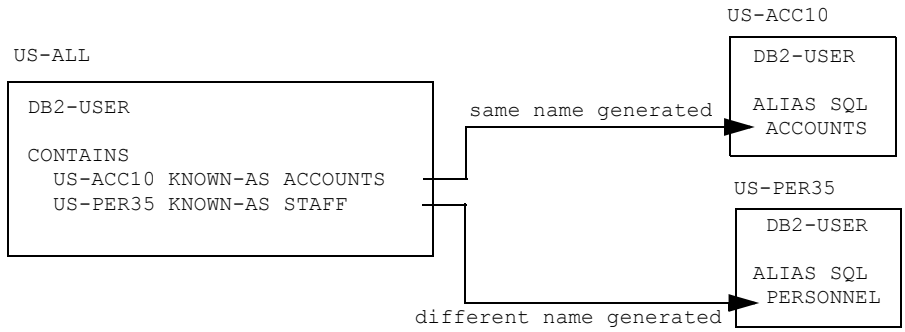
You can optionally define the owner name known by DB2, where it is different from the member name.

To define the DB2 owner name, enter:

```
KNOWN-AS local-name
```

where *local-name* is a creator or owner name, of no more than eight characters, recognized by DB2.

To prevent conflicting definitions, the local name specified in the KNOWN-AS clause should be the same as any SQL ALIAS defined in the DB2-USER to which it refers. For example, in the diagram below the member US-ALL contains two other DB2-USER members, US-ACC10 and US-PER35.



When you generate SQL GRANT or REVOKE statements form US-ALL, the KNOWN-AS clause gives owner names of ACCOUNTS and STAFF. However, when you generate SQL GRANT or REVOKE statements directly from US-PER35, the SQL ALIAS gives a different owner name of PERSONNEL.

If the DB2-USER member, named in the TO clause of a DB2-PRIVILEGE, is a group ID, generated GRANT or REVOKE statements only name the group ID. If the group ID is also defined in the DB2 environment, all the authorization IDs in the group inherit the privileges granted to the group ID, and lose privileges when they are revoked.

However, if you use the USER-EXPANSION keywords available in the DB2 GRANT and DB2 REVOKE commands, the group ID is expanded so that the privilege either applies to the group ID and all the individual user IDs within it, or applies only to the individual user IDs.

Refer to Chapter 8, "Commands," on page 175 for details of the DB2 GRANT and DB2 REVOKE commands.

Defining a User's Synonyms for Aliases, Tables, Views

To define synonyms for the aliases, tables and views of a particular user, enter:

```
SYNONYMS synonym-name FOR object
```

where:

synonym-name is an alternative name, of no more than 18 characters, for the alias, table or view.

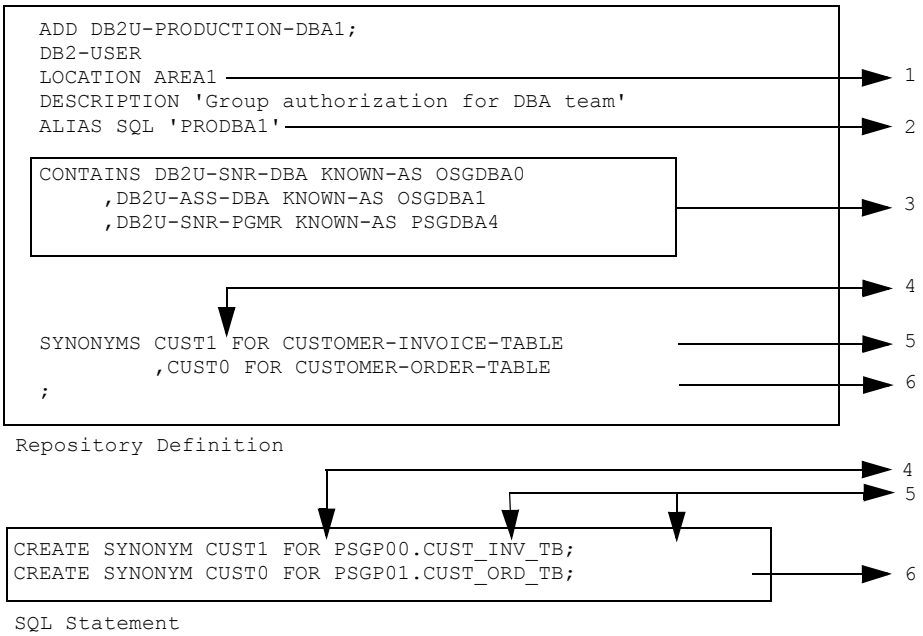
object is the name of a DB2-ALIAS, DB2-TABLE, SQL-TABLE, DB2-VIEW or SQL-VIEW member.

The DB2-synonym for the alias, table or view is taken directly from the name you specify in this clause.

To define several synonyms, each synonym and the object it defines must be separated by commas. For example:

```
SYNONYMS EMP-CODES FOR TA-EMP7C
         , TAX-RECS for VW-TAX9N
```

Example



1 The LOCATION clause sets up a relationship to the DB2-LOCATION member AREA1. This is used to generate three-part qualified names for tables and views owned by DB2U-DBA-PRODUCTION-DBA1.

2 The user's SQL ALIAS, PRODDBA 1 documents the actual authorization ID in the repository, but is not used to generate this SQL statement.

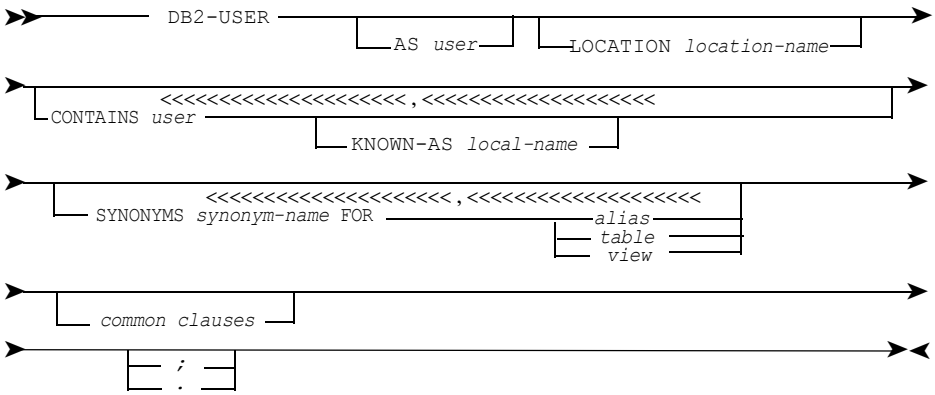
3 The CONTAINS clause documents, the DB2-USER members in the group ID. Local names are defined for convenience and are the same as the SQL ALIASes in the corresponding DB2-USER members.

4 The CREATE SYNONYM statements have to be executed in DB2 by the authorization ID PRODDBA1 to reflect the creator of the synonyms. The first CREATE SYNONYM statement is generated with CUST1 as the synonym name, taken exactly as entered in the repository definition.

5 The two-part qualified table name is derived by taking the SQL ALIAS (PGP00) of the DB2-USER member referred to in the CREATOR-OWNER clause of the DB2-TABLE member CUSTOMER-INVOICE-TABLE. Neither of these two members are shown here.

6 The second CREATE SYNONYM statement is generated in the same way as described in 4 and 5 above.

DB2-USER Syntax



where:

user is the name of a DB2-USER or SQL-USER member.

location-name is the name of a DB2-LOCATION member.

user is as defined above.

local-name is a local name, of no more than 18 characters.

synonym-name is a synonym, of no more than 18 characters.

alias is the name of a DB2-ALIAS member.

`table` is the name of a DB2-TABLE or SQL-TABLE member.

`view` is the name of a DB2-VIEW or SQL-VIEW member.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of `common clauses`.

DB2-VIEW

DB2-VIEW defines DB2 views in the repository.

Refer to "DB2-VIEW Syntax" on page 506 for the syntax of the DB2-VIEW member definition.

Views are of major interest to the end-user, since they provide access to specific data, selected from one or more tables or views. Therefore, with DB2-TABLE, the DB2-VIEW is one of the most used DB2 member types.

To define a DB2 view, enter:

```
DB2-VIEW
```

The member definition must begin with this member type identifier.

All other clauses available to define DB2-VIEW members are optional. However, for the successful generation of SQL statements you must define specific clauses, as follows.

- For CREATE VIEW statements define the CREATOR-OWNER and COLUMNS clauses
- For COMMENT ON statements define the CREATOR-OWNER and DB2-COMMENT clauses
- For DECLARE VIEW statements define the CREATOR-OWNER and COLUMNS clauses
- For DROP VIEW statements define the CREATOR-OWNER clause
- For LABEL ON statements define the CREATOR-OWNER and DB2-LABEL clauses

To specify the ITEM and GROUP members that represent the columns of the view, see the CONTAINS clause. As for DB2-TABLE members, it establishes relationships between a DB2-VIEW and the ITEM and GROUP members that define the view's columns.

You can define columns:

- Individually, so that one ITEM or GROUP member defines one column
- In sets, so that the same ITEM or GROUP member defines several columns, with identical attributes
- In cascades from a GROUP member, so that every ITEM nested in a GROUP member defines one column

Sub-clauses within the CONTAINS clause enable you to define:

- The names of columns
- Calculated values to be held by columns
- That columns are GROUP BY columns
- The table that the columns are derived from
- Associated column comments and labels

The DB2-VIEW member type has specific clauses that enable you to define the tables or views on which the view is based and to define a SELECT clause with optional FROM, WHERE, and HAVING clauses.

DB2-VIEW repository definitions can be generated automatically if you use the Workbench Design Area facilities (WBDA) for DB2 database design.

Refer to Chapter 3, "DB2 Database Design," on page 27 for details of generating DB2-VIEW definitions from the WBDA.

Refer to Chapter 4, "Repository Definition," on page 91 for details of derivation of DB2 view names.

Reusing Existing Member Definitions

Refer to "Reusing Existing Member Definitions" on page 509 for details of reusing all or part SQL-VIEW member definitions using an AS clause.

Defining an Owner

To define the owner of a view, enter:

```
CREATOR-OWNER user
```

where *user* is the name of a DB2-USER or SQL-USER member, and represents the authorization ID of the owner of the view.

On encoding, the member specified in the CREATOR-OWNER clause is checked to ensure that it is a DB2-USER or SQL-USER member.

This clause must be present for the successful generation of SQL statements, but it can be overridden by an SQLID clause defined in a DB2 command.

Note:

The DB2-USER or SQL-USER named in the CREATOR-OWNER clause is used to generate user-qualified names for view. If the DB2-USER has a LOCATION clause defined, and the DB2 profile is set to three-part name generation, location and user-qualified names are generated. The location-qualifier can be overridden by a LOCATION clause defined in a DB2 command.

Defining a Comment on a View or Column

To define a comment for a view or column, enter:

```
DB2-COMMENT 'comment'
```

where *comment* is a string of no more than 254 characters, each line of which is within delimiters.

When generated, lines of comment are concatenated to form a single string. To preserve spaces between words, insert a space between the last character of each continuing line and its delimiter.

For example, the DB2-VIEW named MANAGER-NUMBER has an owner of PERSONNEL and the following comment defined:

```
DB2-COMMENT 'This view contains the Manager number of every '
'manager in each department'
```

The following SQL statement can be generated:

```
COMMENT ON VIEW PERSONNEL.MANAGER_NUMBER IS 'This view cont
ains the Manager number of every manager in each department'
```

In this example the word *contains* has been split due to the margins set in the DB2 profile.

This clause must be present for the successful generation of SQL COMMENT ON statements.

Note:

- For columns, the comment definition must follow the CONTAINS clause defining that column or group of columns
- For views, the comment definition should precede the COLUMNS clause that defines the columns of the view.

Defining a Label on a View or Column

To define a label for a view or column, enter:

```
DB2-LABEL 'label'
```

where *label* is a string of no more than 30 characters, within delimiters.

This clause must be present for the successful generation of SQL LABEL ON statements.

Note:

- For columns, the label definition must follow the CONTAINS clause defining that column or group of columns
- For views, the label definition should precede the COLUMNS clause that defines the columns of the view.

Specifying the Form Description that Defines the Data Type of Columns

To specify which form description, defined in an ITEM or GROUP member, is used to generate the data type of the columns in the view, enter one of the following form keywords:

```
ENTERED-AS
HELD-AS
REPORTED-AS
DEFAULTED-AS
```

The form keyword that you define applies to all the ITEMs and GROUPs named in the CONTAINS clause that follows.

For example, a DB2-VIEW member containing the following lines:

```
ENTERED-AS
CONTAINS ITEM1, ITEM2
```

refers to the two ITEM members:

ITEM1

```
ITEM
HELD-AS BINARY 10
ENTERED-AS CHAR 5
REPORTED-AS FLOAT 9
```

ITEM2

```
ITEM
ENTERED-AS DECIMAL 4.2
DEFAULTED-AS FLOAT 7
```

The ENTERED-AS form keyword in the DB2-VIEW definition specifies that the ENTERED-AS form description from both ITEMs is used to define the data type of columns. Therefore the column generated from ITEMI has a data type of CHAR, and the column generated from ITEM2 has a data type of DECIMAL.

If you do not specify a form keyword then the DEFAULTED-AS form description is used. Where the ITEM or GROUP has no DEFAULTED-AS form description defined, then the Manager Products defaults apply. For further information refer to the *ASG-Manager Products Source Language Generation* guide.

Refer to Chapter 4, "Repository Definition," on page 91 for further details of documenting the columns of views and generating data types.

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of ITEM and GROUP member definitions.

Specifying the ITEMs or GROUPs that Define Columns

To specify the ITEM or GROUP members that define columns of this view, enter:

```
CONTAINS member-list
```

where *member-list* is the name of one or more ITEM or GROUP members, separated by commas, each representing a single column.

The CONTAINS clause specifies the columns that comprise the view, and the subselect (SELECT clause) defines the result table that makes up the view. In most cases, the columns of the view are also the columns selected, and you only need to define the CONTAINS clause to define both. However, there can be cases when the columns selected do not match those in the view. This is possible when there are multiple SELECT clauses in the view. To override the CONTAINS clause columns for a SELECT clause, you must use the COLUMNS subclause of the SELECT clause. Any SELECT clause that does not have a COLUMNS subclause will use the columns in the CONTAINS clause. See "Defining Columns for the Subselects and Fullselects" on page 493 for details on the COLUMNS subclause.

On encoding, the members specified in the CONTAINS clause are checked to ensure that they are either GROUPs or ITEMs. Duplicate column names are not permitted by DB2 therefore column names are checked on generation to ensure that no duplicates are present.

Each ITEM can define up to 15 form descriptions. To define which of the form descriptions you want to use, enter:

```
CONTAINS item version
```

where:

item is the name of an item member

version is an integer in the range 1 to 15, and defines the form description version that you want to use.

For example:

```
HELD-AS CONTAINS STOCK-LIST 3
```

defines that the third HELD-AS form description the ITEM member STOCK-LIST is used as the column data type.

When you use the SIZE and RECALCULATE commands, the data type of columns is used to calculate the size of a table.

To define a set of columns with identical attributes, using the same ITEM or GROUP member, enter:

```
CONTAINS (integer) member
```

where:

integer is the number of columns to be derived from the member, within brackets

member is the name of an ITEM or GROUP member.

To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by _l, the second by _2 and so on.

For example:

```
CONTAINS (4) STOCK-LIST
```

generates the four columns STOCK_LIST_1, STOCK_LIST 2, STOCK_LIST_3, and STOCK_LIST_4. The attributes, such as data type, are the same for each of the four columns.

When a column is derived from a GROUP containing several ITEMs, the data type of the column is taken as CHAR. The maximum number of characters allowed in the column is calculated from the combined field lengths of the form descriptions defined in each ITEM. However, where a DB2 command applied to the DB2-VIEW specifies the EXPAND keyword, then each item within a GROUP generates a separate column.

Naming Columns

You can explicitly name a column if you do not want its name to be derived from the ITEM or GROUP name or alias.

To define the name of a column or view, enter:

```
KNOWN-AS local-name
```

where *local-name* is a string of no more than 18 characters.

For example:

```
CONTAINS IT-INCOMING KNOWN-AS STOCK-IN
```

defines that the ITEM member IT-INCOMING generates a view column called STOCK-IN.

If you use the KNOWN-AS clause to name a column set, the local name is duplicated for each column. To resolve duplicated names on generation of an SQL statement, column names are automatically suffixed by an underscore and a number, the first by _1, the second by _2, and so on.

For example:

```
CONTAINS (3) IT-Q1 KNOWN-AS MONH
```

generates three columns from the ITEM member IT-Q1, named MONTH_1, MONTH_2, and MONTH_3.

To define the name of a column in a view, if it is different from the column name in the table, use the COLUMN-NAME clause.

Refer to Chapter 4, "Repository Definition," on page 91 for further details of the generation of column names.

Specifying that Each ITEM contained in a GROUP Defines One Column

If you want each of the ITEMs contained in a GROUP to represent a column, enter:

```
CONTAINS group EXPAND
```

where *group* is the name of a GROUP member.

For example:

```
CONTAINS AREA-DEPOT EXPAND
```

generates four columns because the GROUP, AREA-DEPOT, contains four ITEMS.

However, if the NO-EXPAND keyword is specified in a DB2 command, the EXPAND keyword is overridden and the GROUP generates a single column.

Note:

You cannot define a KNOWN-AS clause with expanded GROUPs. The generation of SQL CREATE statements is unsuccessful if a DB2 command including the EXPAND keyword is applied to a DB2-VIEW whose columns are named by the KNOWN-AS clause. This is because the local name is duplicated for every column generated from the GROUP member.

If the GROUP is nested, that is it contains other GROUPs, each of these is also expanded so that all ITEMs are used to define columns. Nesting can continue to any depth and is only limited by the amount of memory available.

ELSE clauses defined in expanded GROUP members generate a column with a CHAR or VARCHAR data type with a field length equal to the longest overlaid field.

Where a set of columns is derived form an expanded GROUP, each contained ITEM or GROUP is repeated the number of times specified. If a GROUP contains an ITEM with its own repeating factor, the ITEM is also repeated the number of times specified.

For example, a DB2-VIEW defined as:

```
CONTAINS (2) AREA-DEPOT EXPAND
```

where the GROUP, AREA-DEPOT, contains:

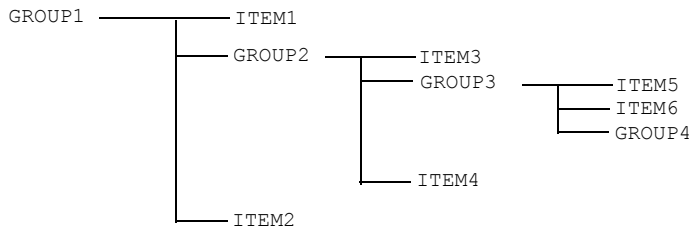
```
(2)      WAREHSE-A
         WAREHSE-B
(3)      LOCALSTK-A
```

generates four columns from WAREHSE-A, two columns from WAREHSE-B, and six columns from LOCALSTK-A, and names them as follows:

```
WAREHSE_A_1
WAREHSE_A_2
WAREHSE_B_1
LOCALSTK_A_1
LOCALSTK_A_2
LOCAKSTK_A_3

WAREHSE_A_3
WAREHSE_A_4
WAREHSE_B_2
LOCALSTK_A_4
LOCALSTK_A_5
LOCALSTK_A_6
```

Expanded GROUPs and Host Language Data Structures

The member GROUP I contains nested GROUPs, shown in the following diagram:



When you generate SQL statements or SQL host language data structures, intermediate levels in the data structure, that is GROUP2 and GROUP3, are removed in order to generate the following flat, two-level structure:

```
02 ITEM1
02 ITEM3
02 ITEM5
02 ITEM6
02 GROUP4
02 ITEM4
02 ITEM2
```

Note:

GROUP4 is treated as an elementary field as it has no lower level. Its data type defaults to CHAR(l).

Intermediate levels, in the above example GROUP2 and GROUP3, can be shown as comments.

When you generate host language data structures in working storage, for example for PL/I, intermediate levels are expanded to give the following nested structure:

```
02 ITEM1,
 02 GROUP2,
      03 ITEM3
      03 GROUP3
         04 ITEM5
         04 ITEM6
         04 GROUP4
      03 ITEM4
02 ITEM2
```

If you consider the original nested structure as a tree and field that do not have lower levels as leaves, then the root of the tree is taken as the first level and only the leaves of the tree are taken as the second level.

There is one exception to this rule: When COBOL data structures are generated, VARCHAR and VARGRAPHIC characters are not treated in this way, since they have a two-level structure anyway.

Refer to the PRODUCE command for details of generating host language data structures.

Defining Column Attributes

You can define additional attributes for a view's column(s) using sub-clauses within the CONTAINS clause. If the CONTAINS clause defines a column set or an expanded GROUP, the generated column attributes apply to all the columns in the set.

To define the name of a column in a view, if it is different from the column name in the table, enter:

```
COLUMN-NAME name
```

where *name* is a name of no more than 18 characters. If you do not define a name for the column, the usual rules for generating column names apply.

If you define a COLUMN-NAME clause for a column set or expanded GROUP, the generated column names are identical, and the statement generation will fail.

To define a calculated value that is held in a column, enter:

```
EXPRESSION 'string'
```

where *string* is an SQL expression, of no more than 255 characters within delimiters, and contains the expression used to calculate the values contained in the column.

When you define an EXPRESSION clause you can generate a column that holds a value calculated by an operation performed on none, one or more of the other columns in the view.

You may rename result columns in a CONTAINS clause by specifying:

```
AS name
```

The ITEM that defines the column should generate a data type compatible with the calculated expression, for SQL DECLARE or PRODUCE statements to be meaningful.

To define that a column is part of a GROUP BY clause for the view, enter:

```
GROUP-BY
```

The keyword has the same meaning as in DB2.

Ambiguity may arise if different columns from different tables or views have the same name. To define a correlation name for a column, so that it can be correlated with the table named in the FROM clause, enter:

```
TABLE correlation-name
```

where *correlation-name* is an identifier, of no more than 18 characters, for the table or view. The same name is repeated in the CORRELATION-NAME clause, where it is associated with the member name of the DB2-TABLE.

For example:

```
CONTAINS
     IT-PERS-13 KNOWN-AS CODE-NAME  TABLE PERSON
,    IT-PROJ-58 KNOWN-AS CODE-NAME  TABLE PROJECT
FROM
     TA-EMP-DETAILS CORRELATION-NAME PERSON
,    TA-PROJ-DATA   CORRELATION-NAME PROJECT
```

defines that:

- The ITEM member IT-PERS-13 generates a column called CODE-NAME
- The ITEM member IT-PROJ-58 generates a column called CODE-NAME
- Columns with a correlation-name of PERSON in the TABLE clause are derived from the DB2-TABLE TA-EMP-DETAILS
- Columns with the correlation name of PROJECT are derived from the DB2-TABLE TA-PROJ-DATA.

To define that a column is to contain a comment enter:

```
DB2-COMMENT 'comment'
```

To define that a column is to contain a label enter:

```
DB2-LABEL 'label'
```

Refer to "Defining a Comment on a View or Column" on page 483 and "Defining a Label on a View or Column" on page 484 for details of how to define comments and labels.

Defining that Updates are Checked

To define that DB2-checks any updates to a view against the view definition, enter:

```
WITH-CHECK-OPTION
```

To define that DB2 checks any updates to a view against the view definition and all underlying views (regardless of whether underlying views are defined with a check option), enter:

```
WITH-CHECK-OPTION CASCADED
```

To define that DB2 checks any updates to a view against the view definition and all underlying views that are defined with a check option, enter:

```
WITH-CHECK-OPTION LOCAL
```

Defining the View Subselect Type

You can define the following clauses in a DB2-VIEW from which subselect clauses in SQL CREATE VIEW statements are generated.

To define that duplicate rows in a view are all preserved, enter:

```
SELECT ALL
```

To define that duplicates are eliminated, enter:

```
SELECT DISTINCT
```

If you do not define a SELECT clause, the SELECT attribute is automatically generated, and the DB2 default applies.

If the SELECT statement is a fullselect that is inside parentheses, enter:

```
SELECT DMS dms-name
```

where *dms-name* is the name of a DB2-DMS member that contains the fullselect statement that is within the parentheses. See "Specifying SELECT Clauses Inside Parentheses" on page 494 for details.

Defining Columns for the Subselects and Fullselects

The SELECT statement in a view creates the result table that comprise the view. This SELECT statement can be a simple statement containing one SELECT clause, or a complex one with more than one SELECT clause.

An example of a simple SELECT statement is:

```
SELECT EMPNO, FIRSTNME, LASTNAME
       FROM EMP
       ORDER BY HIREDATE;
```

Some examples of complex SELECT statements are:

```
SELECT HIREYEAR, SALARY
  FROM (SELECT YEAR(HIREDATE),
               SALARY
         FROM MPRDSN.EMP) AS NEWEMP
  GROUP BY SALARY;
```

where there is a second SELECT clause embedded in the FROM clause of the first SELECT, or

```
 SELECT EMPNO FROM EMP
   WHERE WORKDEPT LIKE 'D%'
   UNION
   SELECT EMPNO FROM EMPPROJACT
      WHERE PROJNO LIKE 'AD%';
```

where there is a second SELECT clause which is unioned with the first SELECT clause. This type of SELECT is not a subselect, but a fullselect.

For a view with a simple SELECT statement with only one SELECT clause, the columns to be selected can be automatically derived from the CONTAINS clause that defines the columns of the view. However, for complex SELECT statements where each SELECT clause may involve different columns, the CONTAINS clause may not be sufficient.

To specify the columns to be selected for a subselect, enter:

```
SELECT option COLUMNS column-list
```

where *option* is either ALL or DISTINCT, and *column-list* is a list of column using these attributes:

- COLUMN-NAME *name*, where *name* is the name of the column selected. *name* must be the member name of an ITEM or its alias that is a column in at least one of the tables named in the FROM clause of this subselect. Either this attribute or the EXPRESSION attribute is required for each column.

If a COLUMN-NAME is present and a correlation name for this column is required, enter:

TABLE *name*

where *name* is the CORRELATION-NAME of the table from which the column is to be selected. *name* must be one of the correlation names in the FROM clause of this subselect.

- EXPRESSION '*expression*'

  expression defines an expression that is used to derive the value of this column. *expression* is mutually exclusive of COLUMN-NAME. Either an EXPRESSION or COLUMN-NAME attribute is required for each column.

  If an EPRESSION is present and a name is to be given this expression, enter:

  AS *name*

  where *name* is the name to be given a column that is defined by this expression.

- GROUP-BY

  This keyword specifies that this column is to be grouped by rows with same values.

Specifying SELECT Clauses Inside Parentheses

In "Defining Columns for the Subselects and Fullselects" on page 493, one of the examples gave an embedded SELECT statement within the FROM clause of another SELECT:

```
CREATE VIEW myview
(HIREYEAR, SALARY)
AS
SELECT HIREYEAR, SALARY
   FROM (SELECT YEAR(HIREDATE) AS HIREYEAR,
                SALARY
            FROM EMP
   GROUP BY SALARY;
```

In such a case, the embedded SELECT clause is coded inside parentheses. In order to enter a SELECT clause that is embedded within a FROM clause, enter:

```
DB2-VIEW
CREATOR-OWNER MPRUSER
HELD-AS
CONTAINS IT-YEAR KNOWN-AS HIREYEAR,
         SALARY
SELECT ALL
  COLUMNS HIREYEAR,
          SALARY GROUP-BY
   FROM from-dms
```

where *from-dms* is the name of a DB2-DMS member that is defined as:

```
DB2-DMS
SELECT ALL
  CONTAINS IT-YEAR KNOWN-AS HIREYEAR
           EXPRESSION 'YEAR(HIREDATE)',
           SALARY
  FROM EMP
```

where IT-YEAR is the name of an ITEM member with the correct data type and size for the expression 'YEAR(HIREDATE)'.

There are other cases where one of more SELECT clauses are inside parentheses. A fullselect can be coded inside parentheses in order to change the execution order of the SELECTs. DB2 executes those SELECT statements inside parentheses first, followed by the rest.

An example of the fullselect within parentheses is:

```
 CREATE VIEW myview
(HIREDATE, SALARY, TERMDATE) AS
SELECT HIREDATE, SALARY
   FROM EMP AS NEWEMP
   GROUP BY HIREDATE
 UNION
   (SELECT HIREDATE, SALARY, TERMDATE
      FROM HISTEMP AS HISTORY
      GROUP BY HIREDATE
    UNION ALL
    SELECT HIREDATE, SALARY
      FROM TEMPEMP AS TEMPORARY
      GROUP BY HIREDATE);
```

To define such a complex SELECT, enter:

```
DB2-VIEW
CREATOR-OWNER MPRUSER
HELD-AS
CONTAINS HIREDATE,
         SALARY,
         TERMDATE
SELECT ALL
  COLUMNS
   COLUMN-NAME HIREDATE GROUP-BY,
   COLUMN-NAME SALARY
  FROM EMP CORRELATION-NAME NEWEMP
UNION
  SELECT DMS union-dms
```

where *union-dms* is the name of a DB2-DMS member that is defined as:

```
DB2-DMS
SELECT ALL
  CONTAINS HIREDATE GROUP-BY,
           SALARY,
           TERMDATE
  FROM HISTEMP CORRELATION-NAME HISTORY
UNION ALL
  SELECT ALL
    CONTAINS HIREDATE GROUP-BY,
             SALARY
    FROM TEMPEMP CORRELATION-NAME TEMPORARY
```

DB2-DMS members used to hold a SELECT statement can also refer to other DB2-DMS members in order to specify further imbedded SELECTs and/or additional levels of nested parentheses, allowing for many levels of parentheses.

Note:

Only SELECT statements that require parentheses require a DB2-DMS member for its definition. Fullselects and subselects not requiring parentheses should be defined directly in the DB2-VIEW definition.

Defining the Tables of a View

To define the tables or view upon which a view is based, enter:

```
FROM member
```

where *member* is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW, or SQL-VIEW member.

For the successful generation of SQL CREATE statements the members named in the FROM clause must include a valid CREATOR-OWNER clause.

To define a correlation name for a table or view that columns are derived from, enter:

```
FROM member CORRELATION-NAME correlation-name
```

where:

member is the name of a DB2-TABLE, SQL-TABLE, DB2-VIEW, or SQL-VIEW member.

correlation-name is an identifier, of no more than 18 characters, for the table or view. The same name is repeated in the TABLE clause.

Defining the Complex Subselect and Join Syntax

FROM-TEXT is an alternative way to specify a complex SELECT statement that is to be used when generating SQL statements. Refer to IBM documentation for the syntax of the SELECT statement. When a SELECT statement cannot be easily defined to the repository, FROM-TEXT allows the statement to be stored as text that is then used to generate SQL statements when export commands such as DB2 CREATE are issued for the view. However, FROM-TEXT contents do not offer support when doing searches for references to DB2 objects that are named within its text.

FROM-TEXT is generated when VIEW definitions are imported from DB2 if the definition contains structures and attributes that are currently not supported in the import process. For example, if the SELECT statement in the VIEW contains a fullselect (multiple subselects unioned together) or if the FROM clause contains joined tables, FROM-TEXT is generated instead of the normal SELECT and FROM clauses.

If there is no FROM-TEXT clause present in the VIEW, the export command DB2 CREATE will generate a FROM statement from the FROM list.

Complex SELECT statements and JOIN tables can be defined in a DB2-VIEW without using FROM-TEXT by editing or creating the member manually rather than through import. The DB2-VIEW definition syntax allows specifying fullselects and joined tables. For details and examples on defining fullselects, see "Defining Columns for the Subselects and Fullselects" on page 493.

A joined table is a result table created by merging 2 or more tables governed by join type and conditions, and can be used in a FROM clause. Specifying a joined table in a FROM clause is very similar in syntax to listing 2 tables, but with the join clause defined on the second table.

To specify a joined table, enter:

```
FROM table-1,
     table-2 JOIN 'type' JOIN-ON 'string'
```

where:

> *table-1* and *table-2* are names of DB2-TABLE members that are being joined. The first table is equivalent to the *left* table in the SQL joined table syntax, and the second table is equivalent to the *right* table.

type is one of these:

- INNER combines each row of both tables minus rows that do not meet the join-on conditions.
- OUTER includes the inner join rows plus the missing rows determined by type of outer join:
  - LEFT OUTER includes rows from the first table missing from the inner join
  - RIGHT OUTER includes rows from the second table missing from the inner join
  - FULL OUTER includes rows from both tables that were missing from inner join

string is the JOIN-ON condition that is from 1 to 254 characters in length. Multiple conditions are allowed for each joined table.

In the above example, *table-1* and *table-2* are joined to produce one result table from which columns are to be selected. If the join clause is omitted:

```
FROM table-1,
     table-2
```

This FROM clause now specifies 2 separate tables from which columns are to be selected.

Defining the WHERE Selection Criteria

To define a where subclause for the subselect, enter:

```
WHERE 'string'
```

where *string* is a valid SQL search condition as defined in the IBM documentation.

Note:

The validity of the search conditions is not checked when a DB2-VIEW member is encoded, or when an SQL statement is generated.

Defining the HAVING Selection Criteria

To define a having subclause for the subselect, enter:

```
HAVING 'string'
```

where *string* is a valid SQL search condition as defined in the IBM documentation.

The validity of the search conditions is not checked when a DB2-VIEW member is encoded, or when an SQL statement is generated.

Example of a DB2-VIEW Definition Containing a JOIN

Two tables, PARTS and PRODUCTS, are joined on the product number (PROD) column to get a table of parts with their suppliers and the products that use the parts. This can be done with the use of the WHERE clause, or the join clause:

```
SELECT PART, SUPPLIER, PARTS.PROD, PRODUCT
 FROM PARTS, PRODUCTS
 WHERE PARTS.PROD = PRODUCTS.PROD;
```

Or

```
SELECT PART, SUPPLIER, PARTS.PROD, PRODUCT
 FROM PARTS INNER JOIN PRODUCTS
 ON PARTS.PROD = PRODUCTS.PROD;
```

To define the view for the first SELECT, enter:

```
DB2-VIEW
  CONTAINS PART, SUPPLIER, PROD TABLE PA, PRODUCT
  AS SELECT ALL
     FROM PARTS CORRELATION-NAME PA, PRODUCTS
     WHERE 'PARTS.PROD = PRODUCTS.PROD'
```

To define the view for the second SELECT, enter:

```
DB2-VIEW
  CONTAINS PART, SUPPLIER, PROD TABLE PA, PRODUCT
  AS SELECT ALL
     FROM PARTS CORRELATION-NAME PA,
           PRODUCTS JOIN INNER JOIN-ON 'PARTS.PROD = PRODUCTS.PROD'
```

In the first view definition, the condition of the row selection from the 2 tables are spelled out in the WHERE clause whereas in the second view definition, the condition is spelled out as a join condition. Both will result in the selection of the same columns for the view. With the first view definition, the rows are selected from 2 separate tables named in the FROM clause, using the WHERE condition to select the rows. With the second view, all rows are selected from a single result table created after the two tables are joined using the join condition.

Example of a DB2-VIEW Definition Containing a GROUP BY, and Generated SQL CREATE Statement for a View Containing a GROUP BY

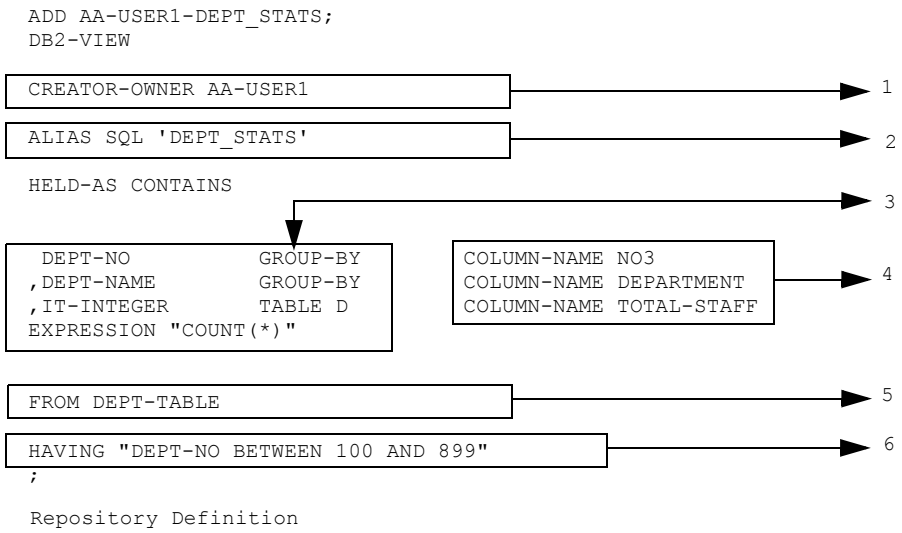
In the following example a view is defined with columns of department number, department name and the total number of employees in the department, for department numbers in the range 100 to 899. The information is derived from a single table: DEPT-TABLE, which contains the columns:

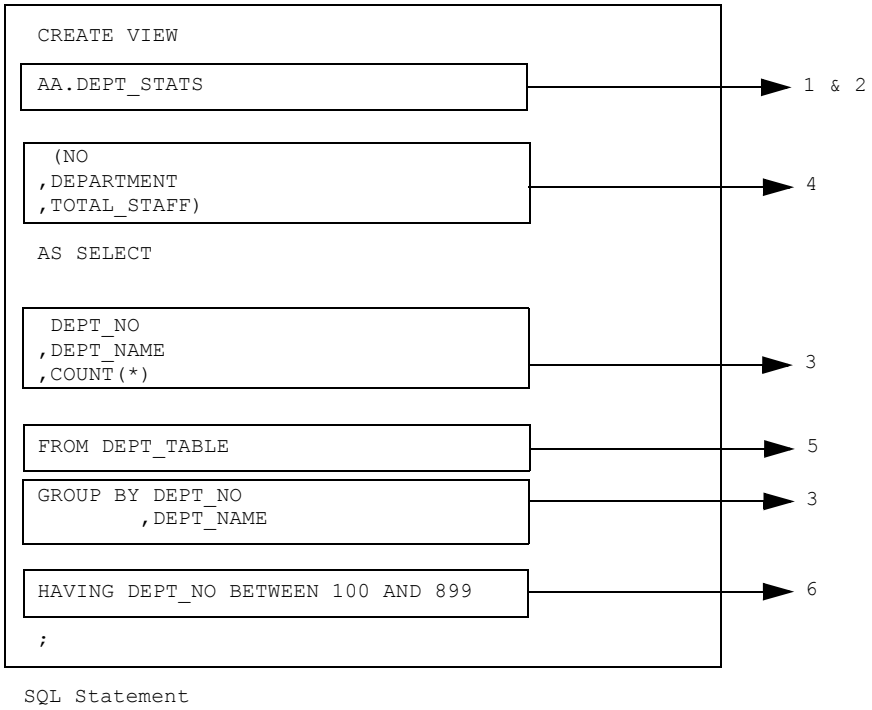
```
     DEPT-NO  DEPT-NAME    EMP-NO
```

The view has 3 columns:

```
NO  derived from DEPT-NO
DEPARTMENT   derived from DEPT-NAME
TOTAL-STAFF  derived by counting the number of rows for each
              department.
```

The view must be defined using GROUP-BY and HAVING clauses, so that the column function COUNT(*) can be used to add up the total number of rows for each department.





1 The derived name for the view is the SQL ALIAS of the DB2-VIEW member, qualified by the SQL ALIAS defined in the DB2-USER member which represents the owner's DB2 authorization ID.

2 Same as step 1.

3 The columns to be selected are taken directly from the CONTAINS clause. The third column, which contains the result of the operation carried out on the DEPT-NO and EMP-NO columns, is defined as an ITEM, IT-INTEGER.

4 The names which the columns in the view are to have are taken directly from the COLUMN-NAME clauses in the member definition.

5 The table from which the columns are drawn is taken directly from the FROM clause in the member definition.

6 The HAVING search condition is taken directly from the HAVING clause in the member definition.

Example of a DB2-VIEW Definition Containing WHERE, HAVING, and GROUP-BY

In the following example, a view is defined with columns containing department number, department name, and total hours worked on the various projects for the department, provided that total exceeds 100 hours. The view must be defined to select rows from the tables DEPT-TABLE (correlation-name D), which contains the columns:

DEPT-NO
DEPT-NAME
PROJ-NO

and PROJ-TABLE (correlation-name P), which contains the columns:

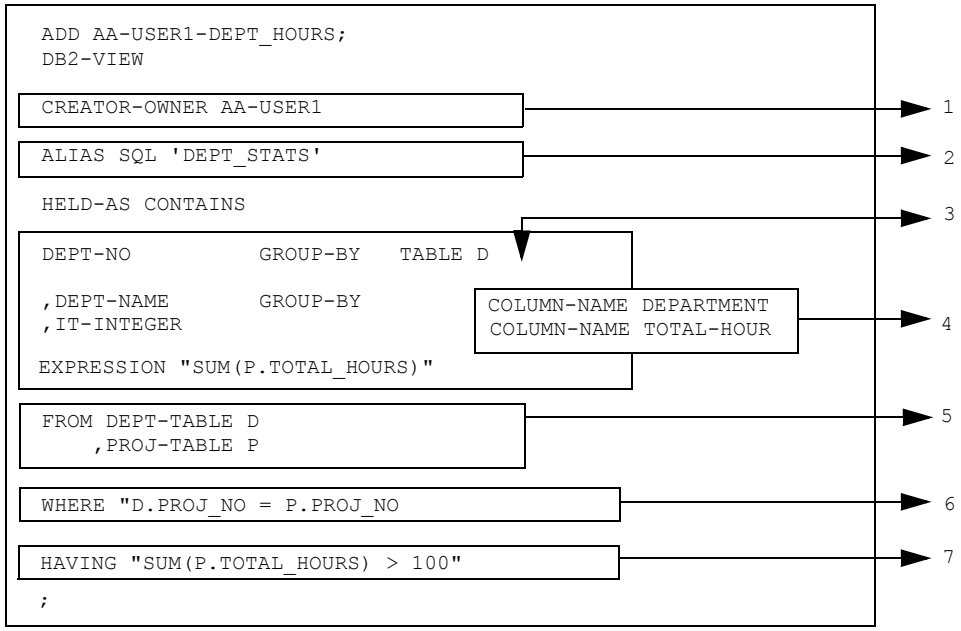
PROJ-NO
PROJ-NAME
TOTAL-HOURS

The view has 3 columns:

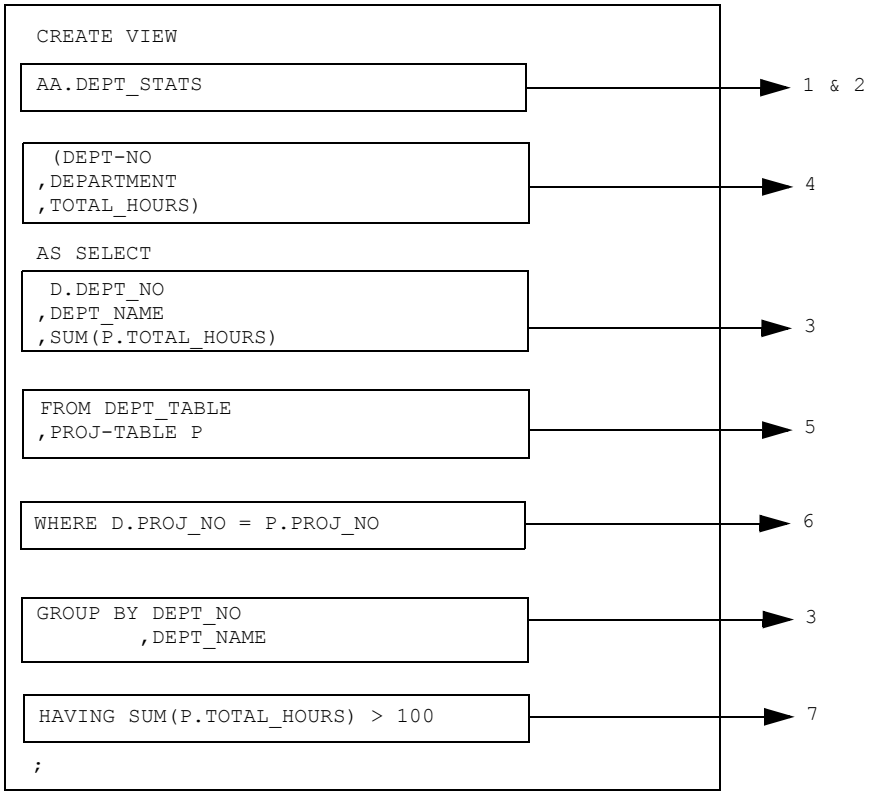
DEPT-NO	derived from DEPT-NO in DEPT-TABLE
DEPARTMENT	derived from DEPT-NAME, in DEPT-TABLE
TOTAL HOURS	derived from the sum of both PROJ-NO columns, which is the join column of DEPT-TABLE and PROJ-TABLE.

The row selection is determined by PROJ-NO, which is present in both tables. The subselect must also have:

- A WHERE clause to select rows from the two tables with matching PROJ-NO values.
- A HAVING clause, to exclude all the groups that do not have total hours exceeding 100 hours.
- A GROUP-BY keyword to sum the total hours in one department.



Repository Definition



SQL Statement

1. The derived name for the view is the SQL ALIAS of the DB2-VIEW member, qualified by the SQL ALIAS defined in the DB2-USER member which represents the owner's DB2 authorization ID.

2. Same as above.

3. The columns to be selected are taken directly from the CONTAINS clause. The third column, which contains the result of the operation carried out on the TOTAL-HOURS columns of DEPT-TABLE and PROJ-TABLE is defined as an ITEM, IT-INTEGER.

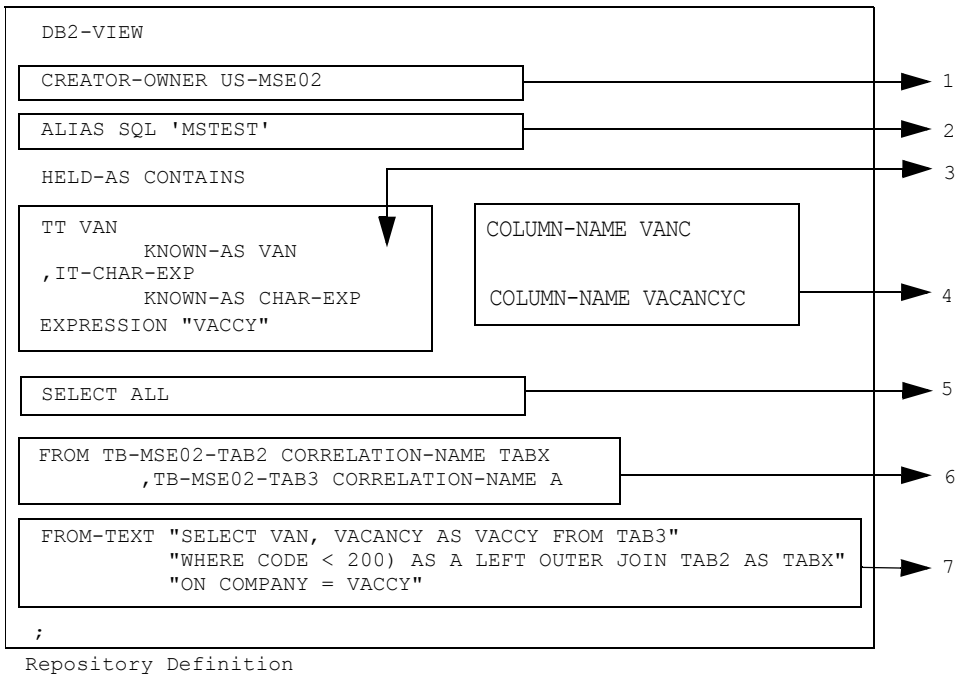
4. The names which the columns in the view are to have are taken directly from the COLUMN-NAME clause in the data definition, or, in the case of the first column, directly from the repository member name of the ITEM which represents the column.

5. The table from which the columns are drawn is taken directly from the FROM clause in the member definition.

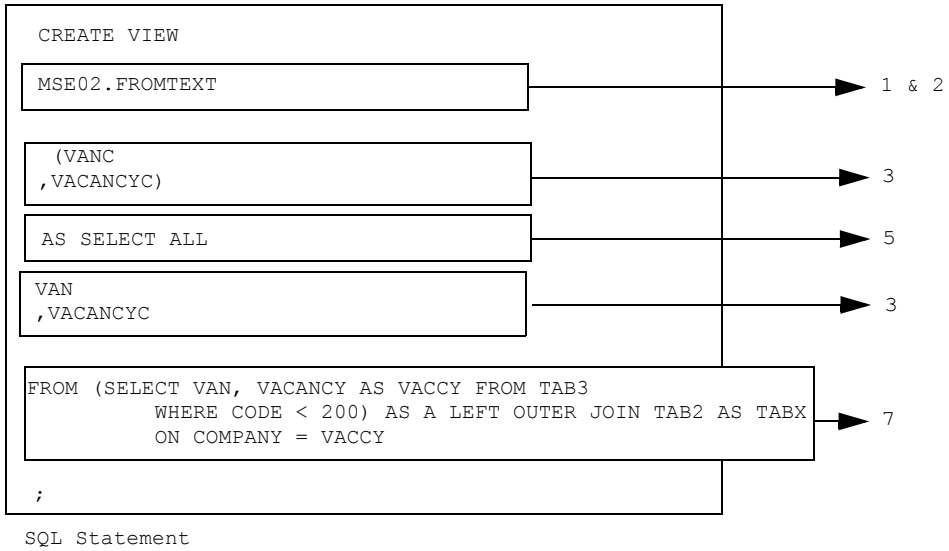
6. The WHERE search condition is generated directly from the WHERE clause.

7. The HAVING search condition is generated directly from the HAVING clause.

Example of a DB2-VIEW Definition Containing FROM-TEXT and Generated SQL CREATE Statement for a View Containing FROM-TEXT



Repository Definition



SQL Statement

1 The derived name for the view is the SQL ALIAS of the DB2-VIEW member, qualified by the SQL ALIAS defined in the DB2-USER member which represents the owner's DB2 Authorization ID.

2 Same as step1.

3 The columns to be selected are qualified by the correlation name of the tables to which they belong.

4 The column names in the view are taken directly from the COLUMN-NAME clause in the member definition.

5 The SELECT option is taken directly from the member definition.

6 The FROM clause in the member definition is generated during import by stripping the table names from the SELECT statement following the DB2 FROM statement. If you are creating your own member definition, the FROM clause should list the table names only and the FROM-TEXT clause should detail the SUB-SELECT statement (if present).

7 The generated FROM statement is taken directly from the FROM-TEXT clause in the member definition.

DB2-VIEW Syntax

```
>>-DB2-VIEW-+-------------+-+-------------------+-+-------------------+-
            +-AS view-----+ +-CREATOR-OWNER user-+ +-DB2-COMMENT 'comment'-+

>-+-------------------+-+-------------+-+-------------------------+-
  +-DB2-LABEL 'label'-+ +-ENTERED-AS--+ +-<<<, <<<-----------------+
                        +-HELD-AS----+ +-CONTAINS columns--------+
                        +-REPORTED-AS+
                        +-DEFAULTED-AS+

>-+-WITH-CHECK-OPTION-+-----------+-+-SELECT-+-------+-+
  +                   +-CASCADED--+ +        +-ALL---+ +
  +                   +-LOCAL-----+ +        +-DISTINCT+

>-+-select-statement-+-----------------------------------+-+
  +                  +-<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<+ +
  +                  +-UNION-+-------+-select-statement-+ +
  +                  +       +-ALL---+                  +

>-+-common-clauses-+-+-;-+-><
  +                  +-.-+
```

where:

view is the name of a DB2-VIEW member.

user is the name of a DB2-USER member.

correlation-name is a string of 1 to 18 characters.

text is a string of up to 32767 delimited character strings, each string having a maximum of 246 characters.

comment is a character string of no more than 254 characters, within delimiters.

label is a character string of no more than 30 characters, within delimiters.

columns is:

```
>-+-+-(n)-+-+-item-+-+-version-+-+-+-KNOWN-AS local-name-+-+
  + +     + +      +-group-----+ +                       +
  +-group EXPAND-------------------------------------------+

>-+-+-COLUMN-NAME name-+-+-+-<<<<<<<<<<,<<<<<<<<<<-+-+
  + +                  + +-EXPRESSION 'expression'-+ +
  +                    +                           +-AS expname-+

>-+-+-GROUP-BY-+-+-+-TABLE correlation-+-+
  + +            + +                   +

>-+-+-DB2-COMMENT 'comment'-+-+-+-DB2-LABEL 'label'-+-+
  + +                       + +                     +
```

(*n*) is the number of columns in a column set.

item is the name of an ITEM member.

version is an integer in the range l to 15.

group is the name of a GROUP member.

local-name is the name of the column in the table and consists of no more than 18 characters.

name is the name of the column in the view and consists of no more than 18 characters.

expression is an expression of no more than 254 characters, within delimiters.

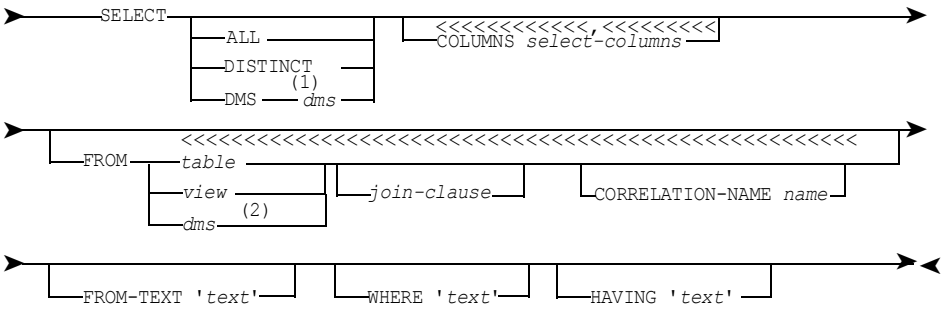
expname is an undelimited name of 1 to 18 characters to be given a column defined by an expression.

correlation is the name of a correlation and consists of no more than 18 characters.

comment is defined above.

label is defined above.

select-statement:


SELECT
ALL
DISTINCT
DMS
(1)
dms
<<<<<<<<<<<<<,<<<<<<<<<<
COLUMNS select-columns
<<<<<<<<<<<<

dms (2) is the name of a DB2-DMS member that holds the definition of a SELECT statement that is embedded within the FROM clause.

join-clause is shown in the syntax below.

name is the correlation name that is to be assigned this table.

text is a string of 1 to 254 characters.

select-columns is:

```
➤──┬─COLUMN-NAME name ──┬──────────────────────┬──┬──────────────────┬──➤
   │                    └─TABLE correlation ───┘  │  └─GROUP-BY ───────┘
   │ <<<<<<<<<<<,<<<<<<<<<<<                      │
   └─EXPRESSION 'expression' ──┬──────────────┬───┘
                               └─AS expname ──┘
```

correlation is defined above.

expression is defined above.

expname is as defined above.

join-clause is:

```
                                          <<<<<<<<<<<<<<<<<
➤──────JOIN ──┬─'INNER' ──────────────────JOIN-ON 'string' ──────────────➤
              ├─'OUTER' ──────────┤
              ├─'LEFT OUTER' ─────┤
              ├─'RIGHT OUTER' ────┤
              └─'FULL OUTER' ─────┘
```

The JOIN clause is specified on the second FROM element. For example,

```
FROM  ...
    ; table-a
    ; table-b JOIN 'INNER'...
      ...
```

Refer to *ASG-Manager Products Dictionary/Repository User's Guide* for details of common clauses.

Reusing Existing Member Definitions

You may wish to generate SQL statements for different DB2 objects that are similar to one another, or are even duplicates. Rather than fully defining several members with the same clauses, you can fully define one member and reuse its clauses.

To reuse clauses already defined in another repository member, enter:

```
As member
```

where *member* is the name of a repository member.

Members named in the AS clause usually have the same member type as the member you are defining. However, if you have SQL/DS support, the following member types can also refer to corresponding SQL members:

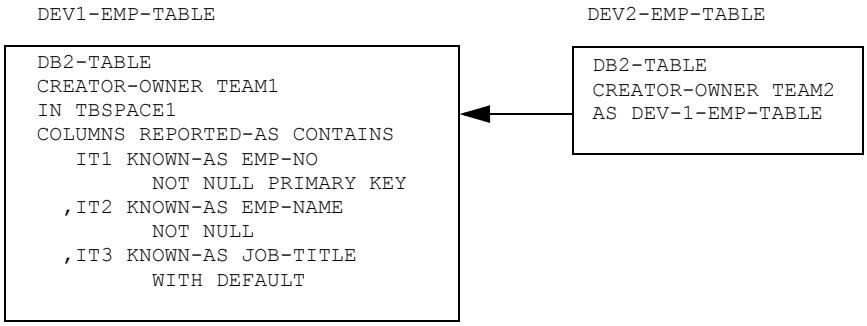
- DB2-INDEX
- DB2-PRIVILEGE
- DB2-TABLE
- DB2-USER
- DB2-VIEW

This enables you to:

- Make use of existing member definitions
- Avoid rekeying data
- Save space in the repository
- Share member definitions across relational databases.

During generation of SQL statements any required clauses that are undefined in the member being generated are taken from the member named in the AS clause.

For example, if different development teams require versions of the same table, you can fully define one member, DEV1-EMP-TABLE, then use the AS clause in the second member, DEV2-EMP-TABLE.



The generated tables both belong to TBSPACE1 and both have the same columns.

When you generate an SQL CREATE statement for DEV2-EMP-TABLE, the CREATOR-OWNER clause in DEV1-EMP-TABLE is not extracted via the AS clause because DEV-EMP-TABLE has its own CREATOR-OWNER clause defined.

Appendix A
Name Reduction Process

When you:

- Generate Database Definition Language (DDL) statements from repository members
- Generate host language data structures in COBOL, Assembler, or PL/I from repository members
- Populate the repository with members generated from the Workbench Design Area (WBDA)

the name reduction process is invoked automatically to ensure that the length of the names generated for the relevant external environment are no longer than the maximum acceptable to that environment.

The principle of the name reduction process is to recognize constituent parts of names and to shorten each part, wherever possible. As a result, duplicate names are less likely to arise than if a simple process of truncation were applied to the complete name, and as much as possible of the meanings of the names is preserved.

The Procedures Language provides the REDUCE function so you can define the parameters for name reduction in other circumstances.

Description

The name reduction process first checks whether the name is a single word (that is, one character string without separator characters). If such a single-word name is longer than permitted by the external environment, the first of the following processes that gives the desired result occurs:

- Single-word names with 15 or less characters are truncated from the right, until the permitted maximum number of characters is achieved
- Single-word names with more than 15 characters have characters removed from the middle, until the permitted number of characters is achieved.

For names that consist of two or more constituent parts, separated by recognized separator characters (such as hyphens or underscores), the name reduction process is as follows:

- If necessary, the longest constituent part of the name is truncated from the right, back to the next character that is not a vowel
- If, after this, the name is still longer than permitted, then the next longest constituent part of the name is truncated from the end, in the same way. This process continues until the name length, including the separator character(s), is within the permitted maximum. In this process, no constituent part of the name is truncated to less than two characters.

If the number of constituent parts of a name is greater than the optimum number that would leave at least two characters in each part, separated by recognized separators, then the first of the following processes that gives the desired result occurs:

- The constituent parts of the name are truncated from the right, back to the next character which is not a vowel, as above
- The separators are removed.

If a name cannot be reduced to its permitted maximum length by any of the processes described above, then the required number of characters (including separator characters) are removed from the middle of the name.

Example

A DB2 table, defined in the repository as a DB2-TABLE member, may have defined in it two columns called:

```
SPECIAL-ORDER-DATE-MONTH
```

and

```
SPECIAL-ORDER-DATE-YEAR
```

Simple truncation to 18 characters would produce, in the external environment

```
SPECIAL_ORDER_DATE
```

and

```
SPECIAL-ORDER-DATE
```

The two columns have identical names, which is illegal.

However, the name reduction process reduces the constituent parts of the first name:

```
SPECIAL
   ORDER
      MONTH
```

in turn, to achieve:

```
SPEC_ORD_DATE_MONT
```

and reduces the constituent parts of the second name:

```
SPECIAL
   ORDER
      DATE
         YEAR
```

in turn, to achieve:

```
SPEC-ORD-DATE-YEAR
```

The result is two unique names. The length of each is within the permitted maximum for DB2 column names (18 characters), and the meanings of the names have been preserved.

Appendix B

Documenting Other Relational Databases

Manager Products enable you to document relational databases such as ORACLE, SYSBASE, and INFORMIX.You can define your own member types, to represent objects in a relational database, by tailoring the repository information model. For example, you can define a new set of member types to document an ORACLE environment, based on the DB2 member types.

Refer to the *ASG-Manager Products User Defined Syntax* or *ASG-MethodManager Administration* manual for further details of tailoring the repository information model.

The objects common to all relational databases are tables. The DB2-TABLE member type clauses can document most of the attributes of relational tables. Some, for example OS/2 and OS/400 tables, can be documented using only a subset of. the clauses available. Others, for example ORACLE tables, may need additional clauses to enable you to document all their attributes. However, when you tailor the repository information model, to define a member type based on the DB2-TABLE, you cannot define additional clauses at the column level. Therefore, to enable you to document attributes and relationships for columns within a table, the DB2-TABLE member type has the following clauses that you can use.

The clauses to define additional column attributes are:

```
COL-RELl member COL-ATT1 'string'
COL-REL2 member COL-ATT2 'string'
COL-REL3 member COL-ATT3 'string'
```

where:

member is the name of a repository member

string is a string of up to 254 characters, within delimiters.

Refer to Chapter 9, "Repository Member Types," on page 341 for the documentation of the DB2-TABLE member type and its generic column attributes.

There are several ways to document your relational environments in the repository:

1 Fully define several members of different types, each representing a different relational table, and containing the same or similar data. For example:

```
DB2-TABLE

 IN DB2-TS1

CREATOR-OWNER DJB
COLUMN HELD-AS
 CONTAINS IT-COL1
 FOR-BIT-DATA
 PRIMARY-KEY
FIELDPROC FP12
CONSTRAINT
 NAMED CNST1
FOREIGN-KEY
 IT-COL1
 KNOWN-AS CUST-NO
;
```

DB2 member definition
DB2-EG1

```
ORACLE-TABLE

 TABLESPACE OR-TS1

 CREATOR-OWNER DJB
 COLUMN HELD-AS
  CONTAINS IT-COL1
  FOR-BIT-DATA
  PRIMARY-KEY
FIELDPROC FP12
CONSTRAINT
 NAMED CNST1
FOREIGN-KEY
 IT-COL1
 KNOWN-AS CUST-NO

PCTFREE 20
MAXTRANS 200
;
```

User defined ORACLE
member definition
ORA-EG1

2 Fully define one generic member that contains data common to all the tables. Define data specific to each table, in members of different types, and refer from these to the generic member using an AS clause. For example:

```
GENERIC-TABLE

  CREATOR-OWNER DJB
  COLUMN HELD-AS
   CONTAINS IT-COL1
   FOR-BIT-DATA
   PRIMARY-KEY
  FIELDPROC FP12
  CONSTRAINT
   NAMED CNST1
  FOREIGN-KEY
   IT-COL1
   KNOWN-AS CUST-NO
;
```

GENERIC member definition
GEN-EG2

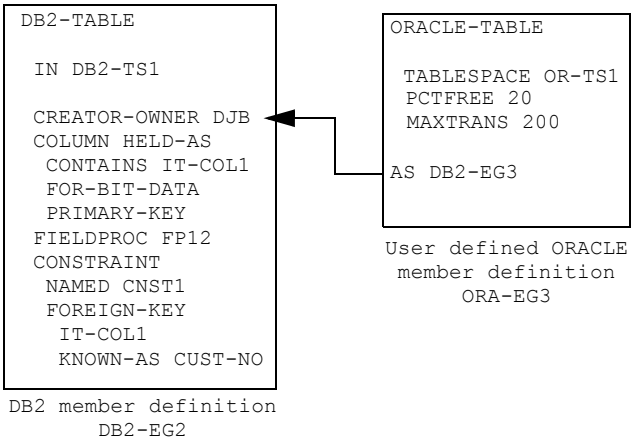
```
DB2-TABLE

 IN DB2-TS1

 AS GEN-EG2
;
```

DB2 member definition
DB2-EG2

```
ORACLE-TABLE

 TABLESPACE OR-TS1
 PCTFREE 20
 MAXTRANS 200

 AS GEN-EG2
;
```

User defined ORACLE
member definition
ORA-EG2

3 Where one of the members is a DB2-TABLE, fully define this as the generic member, and refer from other members representing relational tables using an AS clause. For example:



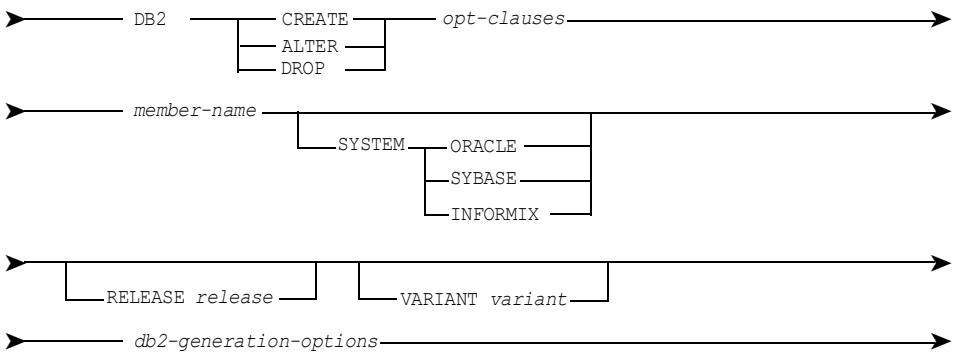
Once you have modeled your database on the repository you may want to update the database using statements generated from repository definitions. If the database uses Structured Query Language (SQL), you can tailor the export to DB2 functions to generate SQL statements that you can then apply to the database.

Refer to Chapter 5, "Export to DB2," on page 105 for details of tailoring export to DB2 functions.

Your repository documentation may become out of step and out of date compared to the database, unless you ensure careful maintenance of the repository definitions.

Refer to the *ASG-DictionaryManager User's Guide* for details of importing from other environments.

To ease the parallel use of other database systems the syntax of the DB2 basic command has been enhanced to:



where:

opt-clauses depends on the database system and are documented in specific Oracle documentation.

member-name is the name of an encoded member that is either based-on a DB2 member type or an additional member type for a specific database system.

release is a valid release-identifier for that database system.

variant is a valid additional identifier.

SYSTEM, RELEASE, and VARIANT are values which are predefined and may be set in the DB2 profile MPDY12PROF. The variables MPDY_DB_SYSTEM, MPDY_DB_RELEASE_NO, and MPDY_DB_SYSTEM_VARIANT contain the respective information during execution of a DB2 generation command.

Appendix C

Defining and Generating DB2 Member Types

This Appendix describes how to define and generate DB2 member types.

Relationship Between DB2 Member Types

The following table shows the relationships that are possible between the member types documenting your DB2 environment.

Table 25 DB2 Member Type Relationships

From Member Type	Via Clause	To Member Type
DB2-ALIAS	AS CREATOR-OWNER FOR	DB2-ALIAS DB2-USER, SQL-USER DB2-TABLE, DB2-VIEW SQL-TABLE, SQL-VIEW
DB2-COLLECTION	AS LOCATION	DB2-COLLECTION DB2-LOCATION
DB2-DATABASE	AS LOCATION STOGROUP	DB2-DATABASE DB2-LOCATION DB2-STOGROUP
DB2-INDEX	AS CONTAINS CREATOR-OWNER ON STOGROUP	DB2-INDEX, SQL-INDEX ITEM, GROUP DB2-USER, SQL-USER DB2-TABLE, SQL-TABLE DB2-STOGROUP
DB2-LOCATION	AS	DB2-LOCATION

Table 25 DB2 Member Type Relationships

From Member Type	Via Clause	To Member Type
DB2-PACKAGE	AS MEMBER COPY QUALIFIER CREATOR-OWNER COLLECTION	DB2-PACKAGE SYSTEM, MMR-SYSTEM, PROGRAM, MODULE DB2-PACKAGE DB2-USER, SQL-USER DB2-USER, SQL-USER DB2-COLLECTION
DB2-PLAN	AS CONTAINS	DB2-PLAN SYSTEM, MMR-SYSTEM, PROGRAM, MODULE, DB2-PACKAGE
DB2-PRIVILEGE	AS CONTAINS GRANTOR ON TO USE TABLESPACE USE STOGROUP	DB2-PRIVILEGE, SQL-PRIVILEGE ITEM, GROUP DB2-USER, SQL-USER DB2-DATABASE DB2-TABLE, DB2-COLLECTION, DB2-PACKAGE, DB2-VIEW,DB2-PLAN DB2-VIEW, SQL-TABLE DB2-USER, SQL-USER DB2-TBSPACE DB2-STOGROUP
DB2-STOGROUP	AS	DB2-STOGROUP
DB2-TABLE	AS CONTAINS CREATOR-OWNER EDITPROC FOREIGN-KEY IN LIKE MEMBER PRIMARY-KEY REFERENCES VALIDPROC	DB2-TABLE, SQL-TABLE ITEM, GROUP DB2-USER, SQL-USER SYSTEM, MMR-SYSTEM, PROGRAM, MODULE ITEM, GROUP DB2-TBSPACE DB2-TABLE, DB2-VIEW SQL-TABLE, SQL-VIEW ITEM, GROUP ITEM, GROUP DB2-TABLE, SQL-TABLE SYSTEM, MMR-SYSTEM, PROGRAM, MODULE

Table 25 DB2 Member Type Relationships

From Member Type	Via Clause	To Member Type
DB2-TBSPACE	AS IN STOGROUP	DB2-TBSPACE DB2-DATABASE DB2-STOGROUP
DB2-PROCEDURE	PARAMETERS INPUTS OUTPUT UPDATES CALLS PASSING PROCESSES CREATOR-OWNER FUNCTION TYPE CAST-FROM COLUMNS TABLE	ITEM, GROUP DB2-PROCEDURE ITEM, GROUP DB2-DMS DB2-USER DB2-PROCDURE ITEM ITEM ITEM DB2-TABLE
DB2-TRIGGER	AS CREATOR-OWNER ON	DB2-TRIGGER DB2-USER DB2-TABLE
DB2-USER	AS CONTAINS SYNONYMS LOCATION	DB2-USER, SQL-USER DB2-USER, SQL-USER DB2-TABLE, DB2-VIEW DB2-ALIAS, SQL-TABLE SQL-VIEW DB2-LOCATION
DB2-VIEW	AS CONTAINS CREATOR-OWNER FROM	DB2-VIEW, SQL-VIEW ITEM, GROUP DB2-USER, SQL-USER DB2-TABLE, DB2-VIEW, SQL-TABLE, SQL-VIEW
SYSTEM, PROGRAM MODULE	CREATOR-OWNER	DB2-USER, SQL-USER
DISTINCT TYPE	NAME	ITEM

Note:

Member typed DB2-TRIGGER is based on the DB2-INDEX for Manager Products Version 2.5. Future releases my supply a base member type. Base Line Technology (BLT) users should add appropriate definitions to their UDS (User Defined System) to add this member type.

Data Types Generated from Form Descriptions

Table 26 Data Types Generated from Form Descriptions

ITEM USAGE Clause	ITEM Form-Description	DATA TYPES BY LANGUAGE: SQL	PL1	COBOL	Assembler
TIME	CHAR t	TIME	CHAR(t)	PIC X(t)	DS Clt
	Other	Accepted but should not be used.			
TIME STAMP	CHAR26	TIMESTAMP	CHAR(26)	PIC X(26)	DS Cl26
	Other	Accepted but should not be used.			
DATE	CHAR d	DATE	CHAR(d)	PIC X(d)	DS Cld
	Other	Accepted but should not be used.			
GRAPHIC	CHAR p p = 1..127	GRAPHIC(p)	GRAPHIC (p)	PIC G(p) DISPLAY-1.	DS CL2p
	CHAR p TO q p = 1..127	VARGRAPHI C (q)	GRAPHIC (q) VAR	10 x. 49 x-L PIC	DS H.CL2q
	CHAR p p > 127	LONG VARGRAPHI C	GRAPHIC (p) VAR	S9(4) COMP. 49 x-D PIC	DS H.CL2p
	CHAR p TO q p > 127		GRAPHIC (q) VAR	G(q/p) DISPLAY-1.	DS H.CL2q
	Other	Error.			
MONEY	Form-description is used as when no USAGE clause present.				
POINTER	Any	Error	Error	Error	Error
ROWID	CHAR(14)	ROWID	CHAR(t)	PIX X(t)	DS CLT
BLOB or CLOB or DBCLOB	CHAR d [K|M|G] d=1...32767	BLOB or CLOB or DBCLOB	CHAR...	PIC...	DS C...
None	BIN p BIN p TO q BIN n,m p, q or n+m = 1..4	SMALLINT	FIXED BIN (15)	PIC S9(4) COMP	DS H

Table 26 Data Types Generated from Form Descriptions

ITEM USAGE Clause	ITEM Form-Description	DATA TYPES BY LANGUAGE: SQL	PL1	COBOL	Assembler
None	BIN p BIN p TO q BIN n,m p, q or n+m = 5..9	INTEGER	FIXED BIN (31)	PIC S9(9) COMP	DS F
None	BIN p BIN p TO q BIN n,m p, q or n+m > 9	FLOAT(21)	FLOAT BIN (21)	COMP-1	DS E
None	DEC n,m n+m = 1..18	DECIMAL (n+m, m)	FIXED DEC (n+m, m)	PIC S9(n)V9(m) COMP-3	DS Plc'a.b'
None	DEC p p = 1..18	DECIMAL(p)	FIXED DEC (p)	PIC S9(p) COMP-3	DS Plc'a'
None None	DEC p TO q q = 1..18 DECp DEC p TO q DEC n,m p, q or n+m = 1..6	DECIMAL(q) Error.	FIXED DEC (q)	PIC S9(q) COMP-3	DS Plc'b'
None	FLOAT p FLOAT p TO q FLOAT n,m p, q or n+m = 1..6	FLOAT(21)	FLOAT BIN (21)	COMP-1	DS E
None	FLOAT p FLOAT p TO q FLOAT n,m p, q or n+m > 6	FLOAT(53)	FLOAT BIN (53)	COMP-2	DS D
None	CHAR p p = 1..254	CHAR (p)	CHAR (p)	PIC X(p)	DS Clp
None	CHAR p TO q p = 1..254	VARCHAR(q)	CHAR (q) VAR	10 x. 49 x-L PIC	DS H,CLq

Table 26 Data Types Generated from Form Descriptions

ITEM USAGE Clause	ITEM Form-Description	DATA TYPES BY LANGUAGE: SQL	PL1	COBOL	Assembler
None	CHAR p p > 254	LONG VARCHAR	CHAR (p) VAR	S9(4) COMP. 49 x-D PIC	DS H,CLp
None	CHAR p TO q p > 254		CHAR (q) VAR	X(q/p).	DS H,CLq
None	Other	Error.			

Instead of an explicit type definition a reference to distinct-types is allowed by:

```
➤──┬──────────────┬──────────────────────────────NAME-item──────────────────────────➤
   │──HELD-AS─────│
   │──REPORTED-AS─│
   │──ENTERED-AS──│
   └──DEFAULTED-AS┘
```

where *item* is the member name of an ITEM (based) member which obeys the syntax to define a DISTINCT TYPE:

```
➤──── member type ──────────────────────────────────────────────────────────────────➤

➤──┬──────────────┬──────────────────────────────format──── DISTINCT-TYPE ──────────➤
   │──HELD-AS─────│
   │──REPORTED-AS─│
   │──ENTERED-AS──│
   └──DEFAULTED-AS┘
```

Explanation

Refer to Chapter 4, "Repository Definition," on page 91 for details of defining form descriptions and generating data types from them.

A PICTURE clause in an ITEM member definition is interpreted as a form- description of CHARACTER *p*, where *p* is the number of bytes that are generated from the PICTURE clause when a record is produced from a member using the export panel or the PRODUCE command. A column with a data type of CHAR *p*, LONG VARCHAR, GRAPHIC *p*, LONG VARGRAPHIC, TIME, TIMESTAMP, or DATE is (depending on the value of *p* and whether the ITEM has a USAGE clause) generated from the ITEM.

Refer to the *ASG-Manager Products Source Language Generation* manual for details of the PRODUCE command.

Columns with data types of VARCHAR(*q*), where *q* is greater than 254, can be generated from ITEMs with:

- A form-description of CHARACTER *p* TO *q* (where *p* is less than 255 and *q* has the desired value)

and without:

- A USAGE clause.

Columns with data types of VARGRAPHIC(*q*), where *q* is greater than 127, can be generated from ITEMs with:

- A form-description of CHARACTER *p* TO *q* (where *p* is less than 128 and *q* has the desired value)

and

- A USAGE clause of GRAPHIC.

where:

p and *q* are integers.

t is a character field length with a default of eight and *d* is a character field length with a default of ten. The systems administrator can tailor the value of *t* and *d* to be compatible with your installation settings for time and date.

Although the USAGE clause and not the form-description is used to generate data types of TIME, DATE, or TIMESTAMP we recommend that you make the form-description match the data type by specifying CHAR t, CHAR d, or CHAR 26.

n indicates the number of decimal digits before the decimal point and *m* indicates the number of decimal digits after the decimal point

x is the column name with underscores changed to hyphens. The data names *x-L* and *x-D* are the result of suffixing *x* with -L or -D and then if necessary the data name is reduced to 30 characters so as not to exceed the COBOL limit for name lengths.

Refer to Appendix A, "Name Reduction Process" on page 511 for details of how names are reduced.

c is the number of bytes required to store the decimal-packed number and can be calculated from one of the following formulas:

- DEC n.m
  c = (n + m + 1)/2
- DEC p
  c = (p + 1)/2
- DEC p TO q
  c = (q + 1)/2

rounded up to the nearest integer.

a is a sequence of nines. The number of nines is equal to the value of *p* or *n*

b is a sequence of nines. The number of nines is equal to the value of *q* or *m*.

NUMERIC-CHARACTER Form-descriptions

NUMERIC-CHARACTER form-descriptions may be used in DB2 column items. They will generate either DECIMAL or CHARACTER data types, depending on the setting of the profile variable MPDY_CM_NUM_GEN. This may be set to :DEC: or :CHAR:. The default is DECIMAL data type generation. The exact format of the data generated is described in "Data Types Generated from Form Descriptions" on page 522 under DEC or CHAR. The data types will be identical to those generated for the selected form-description.

Index

A

Alias
- documenting remote DB2 objects 342

AS clause 509

C

Columns
- documenting data types 94
- generating names 98

Commands 176

conventions page viii

D

Data types
- documented in ITEMs 522
- generating 94
- represented by form description 94

DB2 BIND command 195
- syntax 198

DB2 command output 54
- DB2 LIST CYCLES 83
  - description 83
  - example 84
- DB2 LIST TABLES 81–82
  - description 82
  - example 83
- DB2 PLOT CLUSTER 63
  - cluster diagram 63
  - design relationship matrix 69
  - example 65
- DB2 PLOT REFERENTIAL-STRUCTURES 71
  - additional plots 76
  - example 80
  - layout 72
  - use of directories 77
  - use of pointers 74
- DB2 REPORT 54
  - contents of tables 55
  - example 57
  - foreign key relationships 55

DB2 COMMENT command 206
- syntax 217

DB2 CREATE command 212
- syntax 217

DB2 database design
- DB2 design
  - generating 34
  - populating the dictionary 36
  - reporting 35
- entity and userview models
  - creating 32
- examples 37
  - Department Model 37
  - introduction 37
  - Parts Model 47
- introduction 28
- overview
  - features to support DB2 31
  - referential structures and cycles 31
- relational schemas
  - generating 33

DB2 DATABASE member type 351
- syntax 355

DB2 DEBUG command 219
- syntax 223

DB2 DECLARE command 223
- syntax 226

DB2 DROP command 228
- syntax 232

DB2 GRANT command 234
- syntax 237

DB2 LABEL command 206
- syntax 210

DB2 LIST CYCLES command 239
- syntax 240

DB2 LIST TABLES command 240
- syntax 241

DB2 PLOT CLUSTER command 242
- syntax 244

DB2 PLOT
REFERENTIAL-STRUCTURES
command 245
syntax 250
DB2 POPULATE command 250
syntax 260
DB2 PREVIEW command 262
syntax 272
DB2 PRODUCE command 273
syntax 280
DB2 profile
altering
DATE and TIME character field lengths 115
DB2 subsystem or plan 123
DROP report layout 122
export-generated object name length 123
generated column data type 124
indicator array name suffixes 119
indicator structures 118
indicator suffixes 118
message tolerance level 121
release flag 117
release of DB2 117
space character for SQL DECIMAL datatypes 122
SQL escape character 121
user exits 126
variable-length column name suffixes 119
width of SQL COMMENT ON statements 122
generating
flat or nested data structures 117
host language data structures with SQL DECLARE statements 117
object names and external names from aliases 114
SQL COMMENT ON or LABEL ON statements 119
SQL DECLARE statements with host language data structures 117
two or three-part names 120
introduction 110
summary of variables 110
DB2 REBIND command 195
syntax 202
DB2 RECALCULATE command 282
syntax 284
DB2 RENAME 285
DB2 REPORT command 288
syntax 291
DB2 REVOKE command 234
syntax 237
DB2 SIZE command 291
syntax 294
DB2 SYNONYM command 295
syntax 297
DB2-ALIAS 342
DB2-ALIAS member type 342
syntax 346
DB2-ALTER command 176
syntax 187
DB2-AUXTABLE 347
DB2-COLLECTION member type 350
syntax 351
DB2-DMS member type 355
syntax 358
DB2-FUNCTION 363
DB2-INDEX member type 370
syntax 387
DB2-LOCATION member type 390
syntax 391
DB2-PACKAGE member type 392
syntax 396
DB2-PLAN member type 398
specifying
cache size 401
current server 401
system environment 402
syntax 404
DB2-PRIVILEGE member type 405
syntax 414
DB2-PROCEDURE member type 417
DB2-STOGROUP member type 422
syntax 424
DB2-TABLE member type 425
syntax 450
DB2-TBSPACE member type 456
syntax 467
DB2-TEMPTABLE 469
DB2-TRIGGER member type 472
DB2-USER member type 476
syntax 480
DB2-VIEW member type 481
syntax 506
Default 150
Deriving names
catalogs 99
columns 98
databases 98

distributed objects 101
external objects 97
guidelines 101
indexes 98
passwords 99
programs 99
storage groups 98
synonyms 99
table spaces 98
tables 98
views 98
Distributed network
documenting locations 390
documenting object aliases 342
Documenting
data types 94
DB2 objects 92
DB2 security system 96
different databases 515
Dynamic SQL functions
output 133
security and authorization 133
using Executive Routines
altering a table 134
assigning imported information to Command Variables 144
COMMAND and EXECUTIVE members used 141
creating a table 135
creating and populating a table 141
creating HELP text 147
defining a table 135
importing from tables and views 138
inserting rows in a table 142
submitting prepared SQL statements 146
variables used 139

E

Export to DB2
defining column data types 106
introduction 106
submitting generated output 107
EXTRACT DB2 command 299
syntax 305

F

Form description
representing data types 94

G

Generated member definitions
DB2 POPULATE and DB2 PREVIEW 85
Guidelines for naming objects 101

I

Import from DB2
column documentation 155
introduction 149
naming guidelines 152
tailoring
Corporate Executive Routines 158
Global Variables used 160
Imported objects
default names and types given 150
Interrogation keywords
DB2 member types 103
ISQL command 306
syntax 308

L

Location
documenting 390

M

Member type
documenting the security system 96
identifier 92
relationships 519

N

Name editing options
dropping or replacing a name 338
inserting a string in a name 338
Name reduction process 511
NUMERIC-CHARACTER
Form-description 526

O

Objects in DB2
documenting 342
generating names 97
One-part names 101
Output generation options
examples 336
sending generated data to a USER-MEMBER on the MP-AID 334
sending generated output
to a partitioned data set 336
to a sequential data set 335

P
package/collection variables 169
packagelist entries, expanding 302
POPULATE command 309
    syntax 310
PREVIEW IMPORT command 312
    syntax 314

R
RADD command 316
REBIND 195
REBIND command 195
    syntax 202
RECONCILE command 317
    syntax 328
Re-using existing DB2 member
    definitions 509
RIGN command 329
RREN command 330
RREP command 331
RUPD command 332
    syntax 333

S
Security in DB2 documenting 96
setting switches 130
SQL statements
    generating non-standard 130

T
Three-part names 100
Two-part names 99

U
user exit routines 124
    writing 128
User exits
    for accessing a repository
        member 125
    for one command/panel 129
    for specified export functions 125
    introduction 124