

# **ASG-Manager Products™**

## **Relational Technology Support: SQL/DS**

Version: 2.5

Publication Number: MPR0200-25-RELSQ

Publication Date: November 1996

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.

© 1998-2001 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](http://asg.com)

1333 Third Avenue South, Naples, Florida 34102 USA Tel: 941.435.2200 Fax: 941.263.3692 Toll Free: 1.800.932.5536







# ASG Support Numbers

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. To expedite response time, please follow these procedures.

## **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 or displayed
- A description of the specific steps that immediately preceded the problem
- 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.)

## **If You Receive a Voice Mail Message:**

- 1 Follow the instructions to report a production-down or critical problem.
- 2 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.
- 3 Please have the information described above ready for when you are contacted by the Support representative.

## **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                 |

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.

## ***Business Hours Support***

| <b>Your Location</b>            | <b>Phone</b>  | <b>Fax</b>      | <b>E-mail</b>      |
|---------------------------------|---|-----------------|--------------------|
| <b>United States and Canada</b> | 800.354.3578<br>1.941.435.2201<br><b>Secondary Numbers:</b><br>800.227.7774<br>800.525.7775 | 941.263.2883    | support@asg.com    |
| <b>Australia</b>                | 61.2.9460.0411  | 61.2.9460.0280  | support.au@asg.com |
| <b>England</b>                  | 44.1727.736305  | 44.1727.812018  | support.uk@asg.com |
| <b>France</b>                   | 33.141.028590   | 33.141.028589   | support.fr@asg.com |
| <b>Germany</b>                  | 49.89.45716.300   | 49.89.45716.400 | support.de@asg.com |
| <b>Singapore</b>                | 65.224.3080   | 65.224.8516     | support.sg@asg.com |
| <b>All other countries:</b>     | 1.941.435.2201  |                 | support@asg.com    |

## ***Non-Business Hours - Emergency Support***

| <b>Your Location</b>            | <b>Phone</b>   | <b>Your Location</b>       | <b>Phone</b>       |
|---------------------------------|--|----------------------------|--------------------|
| <b>United States and Canada</b> | 800.354.3578<br>1.941.435.2201<br><b>Secondary Numbers:</b><br>800.227.7774<br>800.525.7775<br><b>Fax:</b><br>941.263.2883 |                            |                    |
| <b>Asia</b>                     | 011.65.224.3080  | <b>Japan/Telecom</b>       | 0041.800.9932.5536 |
| <b>Australia</b>                | 0011.800.9932.5536   | <b>New Zealand</b>         | 00.800.9932.5536   |
| <b>Denmark</b>                  | 00.800.9932.5536   | <b>South Korea</b>         | 001.800.9932.5536  |
| <b>France</b>                   | 00.800.9932.5536   | <b>Sweden/Telia</b>        | 009.800.9932.5536  |
| <b>Germany</b>                  | 00.800.9932.5536   | <b>Switzerland</b>         | 00.800.9932.5536   |
| <b>Hong Kong</b>                | 001.800.9932.5536  | <b>Thailand</b>            | 001.800.9932.5536  |
| <b>Ireland</b>                  | 00.800.9932.5536   | <b>United Kingdom</b>      | 00.800.9932.5536   |
| <b>Israel/Bezeq</b>             | 014.800.9932.5536  |                            |                    |
| <b>Japan/IDC</b>                | 0061.800.9932.5536   | <b>All other countries</b> | 1.941.435.2201     |

## ASG Web Site

Visit <http://www.asg.com>, ASG's World Wide Web site.

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

If you do not have access to the web, FAX your suggestions to product management at (941) 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.



---

# Contents

---

|   |    |
|---|----|
| Preface   | v  |
| About this Publication  | v  |
| Publication Conventions   | vi |
| 1 Introduction  | 1  |
| Overview  | 1  |
| Features  | 2  |
| Introduction  | 2  |
| Corporate Dictionary/Repository   | 2  |
| Design Diagramming Tool   | 3  |
| Database Design Tool  | 3  |
| Data Definition Language (DDL) Generator                                      | 3  |
| COBOL, PL/I, and ASSEMBLER Generator  | 4  |
| Dynamically Submitting SQL Statements to DB2 or SQL/DS                        | 4  |
| Importing Information from SQL/DS   | 5  |
| Functions: How to Use the Tools We Provide                                    | 5  |
| Introduction  | 5  |
| Standards   | 5  |
| SQL/DS Database Design  | 6  |
| Implementation  | 9  |
| Maintenance   | 11 |
| Summary   | 14 |
| Benefits  | 14 |
| Introduction  | 14 |
| A Shared and Re-usable Corporate Model  | 14 |
| Automated Design  | 16 |
| Conclusion  | 17 |
| 2 What Do You Want to Do?   | 19 |
| Introduction  | 19 |
| ASG Support for Your SQL/DS Environment                                       | 19 |
| SQL/DS Database Design  | 19 |
| Producing Output Describing the SQL Design                                    | 19 |
| Generating and Populating SQL Dictionary Definitions of Specified Member Type | 20 |
| SQL/DS Dictionary Definition  | 20 |
| Documenting an SQL/DS Dictionary Schema                                       | 20 |
| SQL/DS Object Definition  | 20 |
| SQL/DS Object Generation  | 21 |

|   |   |    |
|---|---|----|
|   | Generating SQL Statements and SQL/DS Host Language Data Structures                      | 21 |
|   | Generating Tailored SQL Statements and SQL/DS Host Language Data Structures             | 21 |
|   | Dynamically Submitting SQL Statements   | 22 |
|   | Importing Information about SQL/DS Objects  | 22 |
| 3 | SQL/DS Database Design  | 25 |
|   | Introduction to SQL/DS Database Design  | 25 |
|   | Overview  | 25 |
|   | Support for Referential Integrity   | 26 |
|   | Introduction to Referential Structures and Cycles                                       | 27 |
|   | Features to Support SQL/DS  | 28 |
|   | Designing a SQL/DS Database   | 29 |
|   | Creating Entity and Userview Models   | 29 |
|   | Generating a Relational Schema  | 29 |
|   | Generating the SQL Design   | 30 |
|   | Reporting the SQL Design  | 31 |
|   | Populating the Dictionary with SQL Members  | 31 |
|   | Examples of the SQL/DS Database Design Process  | 33 |
|   | SQL/DS Design Analysis  | 48 |
|   | Output from the SQL REPORT Command  | 48 |
|   | Output from the SQL PLOT CLUSTER Command  | 56 |
|   | Output from the SQL PLOT REFERENTIAL-STRUCTURES Command                                 | 64 |
|   | Output from the SQL LIST TABLES Command   | 73 |
|   | Output from the SQL LIST CYCLES Command   | 74 |
|   | Generated SQL Member Definitions  | 76 |
|   | Generated SQL-TABLE Member  | 76 |
|   | Generated SQL-INDEX Member  | 77 |
|   | Generated SQL-VIEW Member   | 78 |
|   | Generated SYSTEM Member   | 79 |
| 4 | Dictionary Definition   | 81 |
|   | Introduction to Documenting an SQL/DS DBMS  | 81 |
|   | Documenting SQL/DS Objects  | 82 |
|   | Clauses Establishing Relationships between SQL/DS Member Types                          | 83 |
|   | Documenting the Columns of SQL/DS Tables and Views                                      | 84 |
|   | Documenting SQL/DS Security Information   | 88 |
|   | Naming Conventions for SQL/DS Members   | 89 |
|   | The Derivation of External Names from SQL/DS Members                                    | 89 |
|   | The Derivation of Column Names from SQL/DS Members                                      | 90 |
|   | The Derivation of the Names of Tables, Views, Indexes, and Dbspaces from SQL/DS Members | 90 |
|   | The Derivation of the Names of Synonyms, Constraints, Correlations, and Programs        | 90 |
|   | The Derivation of SQL/DS User Names   | 91 |
|   | Naming Guidelines for SQL/DS Members  | 91 |
|   | Processing Your SQL/DS Members  | 92 |

|   |  |     |
|---|--|-----|
| 5 | Implementation and Maintenance   | 95  |
|   | Introduction to Generating SQL Statements and SQL/DS Host Language Data Structures                 | 95  |
|   | Overview of Generating SQL Statements and SQL/DS Host Language Data Structures                     | 95  |
|   | Generating Column Data Types   | 96  |
|   | Submitting Generated Output to Your Relational Environment   | 97  |
|   | Tailoring SQL Statements and SQL/DS Host Language Data Structures                                  | 97  |
|   | Introduction to Tailoring  | 97  |
|   | Displaying Internal Diagnostic Output  | 100 |
|   | Generating Object Names and External Names from Aliases  | 101 |
|   | Generating SQL CREATE, LABEL ON, and COMMENT ON Statements from One Member at the Same Time        | 102 |
|   | Generating a Host Language Indicator Structure   | 103 |
|   | Tailoring DATE and TIME Character Field Lengths  | 104 |
| 6 | Generation of SQL/DS Application Programs  | 105 |
| 7 | Dynamic SQL Services   | 107 |
|   | Introduction to Dynamic SQL Services   | 107 |
|   | Overview of Dynamic SQL Services   | 107 |
|   | Security and Authorization   | 107 |
|   | Output Printed by Dynamic SQL Services   | 108 |
|   | Creating Executive Routines to Dynamically Submit SQL Statements to Your DB2 or SQL/DS Environment | 109 |
|   | Introduction to Dynamically Submitting SQL Statements from within Executive Routines               | 109 |
|   | Variables Used in Dynamic SQL Services   | 113 |
|   | The COMMAND and EXECUTIVE Members Used in Dynamic SQL Services                                     | 115 |
|   | Creating and Populating a Table  | 115 |
|   | Inserting Rows into a Table  | 117 |
|   | Importing Information and Assigning it to Command Variables  | 118 |
|   | Submitting any SQL Statement That Can be Prepared  | 120 |
|   | Creating Your Own HELP Text  | 122 |
| 8 | Import   | 123 |
|   | Introduction to Importing Information about External Objects                                       | 124 |
|   | Overview of Importing Information  | 124 |
|   | Naming Guidelines When Importing Information   | 125 |
|   | How Columns Are Documented   | 126 |
|   | Tailoring Import Commands  | 128 |
|   | Introduction to Tailoring Import Commands  | 128 |
|   | Tailorable Corporate Executive Routines  | 130 |
|   | Global Variables Used in the Import Commands   | 131 |
| 9 | Member Types and Commands  | 143 |
|   | Member Type and Command Descriptions   | 143 |
|   | EXTRACT SQL  | 143 |

|  |     |
|--|-----|
| ISQL                                     | 149 |
| POPULATE                                 | 151 |
| PREVIEW                                  | 153 |
| RADD                                     | 156 |
| RECONCILE                                | 157 |
| RIGN                                     | 168 |
| RREN                                     | 169 |
| RREP                                     | 170 |
| RUPD                                     | 171 |
| SQL ACQUIRE                              | 172 |
| SQL ALTER                                | 173 |
| SQL COMMENT and SQL LABEL                | 180 |
| SQL CREATE                               | 182 |
| SQL-DBSPACE                              | 184 |
| SQL DROP                                 | 187 |
| SQL GRANT and SQL REVOKE                 | 190 |
| SQL-INDEX                                | 192 |
| SQL LABEL                                | 197 |
| SQL LIST CYCLES                          | 197 |
| SQL LIST TABLES                          | 198 |
| SQL PLOT CLUSTER                         | 200 |
| SQL PLOT REFERENTIAL-STRUCTURES          | 203 |
| SQL POPULATE                             | 208 |
| SQL PREVIEW                              | 218 |
| SQL-PRIVILEGE                            | 229 |
| SQL PRODUCE                              | 236 |
| SQL REPORT                               | 240 |
| SQL REVOKE                               | 242 |
| SQL SIZE                                 | 243 |
| SQL SYNONYM                              | 244 |
| SQL-TABLE                                | 245 |
| SQL-USER                                 | 259 |
| SQL-VIEW                                 | 262 |
| Defining an AS Clause                    | 277 |
| Filing Generated Output in a User-member | 278 |

Appendix

|   |     |
|---|-----|
| The Manager Products Name Reduction Process | 279 |
| Introduction to the Name Reduction Process  | 279 |
| Description of the Name Reduction Process   | 279 |
| Example of Name Reduction                   | 280 |

Glossary 283

Index 289

---

# Preface

---

The *ASG-Manager Products Relational Technology Support: SQL/DS* manual describes the support provided by the ASG-Manager Family of Program Products for the SQL/DS relational environment. It is assumed that you have basic understanding of ASG-Manager Products (herein called Manager Products) and the computing industry in general, and you are familiar with the SQL/DS environment and SQL/DS technology.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on the Manager Products.

## About this Publication

The *ASG-Manager Products Relational Technology Support: SQL/DS* consists of these chapters:

- Chapter 1, "Introduction," gives you an introductory overview of the support provided by the Manager Products Family for SQL/DS.
- Chapter 2, "What Do You Want to Do?," directs you to the relevant documentation in this manual.
- Chapter 3, "SQL/DS Database Design," describes how to automate the production of a first cut SQL/DS database design and how to populate your Corporate Dictionary/Repository with the relevant member definitions.
- Chapter 4, "Dictionary Definition," describes how to document an SQL/DS environment in your Corporate Dictionary/Repository.
- Chapter 5, "Implementation and Maintenance," describes how to use Manager Products to implement and maintain your SQL/DS environment and how to generate COBOL, PL/I, and Assembler data descriptions from your dictionary definitions. Sections dealing with tailoring may have been removed by the Systems Administrator.
- Chapter 6, "Generation of SQL/DS Application Programs," will be issued in a forthcoming Amendment List. It will describe the use of Manager Products in the generation of SQL/DS application programs.

- Chapter 7, "Dynamic SQL Services," describes how to dynamically submit SQL statements to SQL/DS, and receive the results, from within Manager Products. Sections dealing with tailoring may have been removed by the Systems Administrator.
- Chapter 8, "Import," describes how to import information about SQL/DS objects and how to use the information to populate the Corporate Dictionary/Repository with member definitions. Sections dealing with tailoring may have been removed by the Systems Administrator.
- Chapter 9, "Member Types and Commands," brings together the specifications of all SQL/DS-related Manager Products commands and MEMBER types, arranged alphabetically. After you have familiarized yourself with the concepts of Manager Products support of SQL/DS and the facilities offered, as described in Chapters 1 through 8, this chapter will thus provide a directly accessible source for subsequent technical reference. We assume that you understand SQL/DS and its terminology.

## Publication Conventions

The following conventions apply to syntax diagrams that appear in this manual.

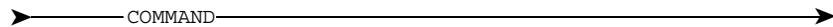
Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

| <b>Convention</b>   | <b>Represents</b>   |
|---|---|
|  | 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 is continues from the line above. |

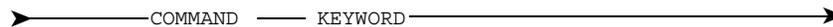
Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

Statement identifiers appear on the main path of the diagram:



A required keyword appears on the main path:



An optional keyword appears below the main path:





Allen Systems Group technical publications use these conventions:

| <b>Convention</b>                     | <b>Represents</b>   |
|---------------------------------------|---|
| 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).    |
| <i>lowercase italic<br/>monospace</i> | Information that you provide according to your particular situation. For example, you would replace <i>filename</i> 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.<br><br>Also used for denoting brief examples in a paragraph. |

---

# 1

## Introduction

---

### Overview

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.

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

Using our tools you can build and maintain an efficient DB2 or SQL/DS environment far faster than is possible using manual methods.

The tools are based upon use of a Corporate Dictionary/Repository in which you can store all of your organization's documentation and program code. They share a common user interface (of which online documentation and HELP are part) which you can tailor to suit the environment you are used to and to suit users with differing skills.

Using our import facilities you can populate the Corporate Dictionary/Repository with members generated from information imported from DB2 and SQL/DS.

We assume that you understand DB2 and SQL/DS and their terminology.

## **Features**

### **Introduction**

We provide the following tools to support DB2 and SQL/DS system life cycles:

- A Corporate/Dictionary Repository
- A Design Diagramming Tool
- A Database Design Tool
- A Data Definition Language (DDL) generator
- A COBOL, PL/I, and ASSEMBLER data structure generator
- Dynamic SQL Services
- Import facilities.

We recommend that you use all of the above to support the entire system life cycle. However, you can use each tool independently, as suits your purpose.

### **Corporate Dictionary/Repository**

The following list highlights the features of the Corporate Dictionary/Repository:

- A repository in which to store all of your documentation and program code
- Supports documentation of all DB2 and SQL/DS components including security information and DB2 plans
- Supports Referential Integrity
- All major DBMS, including IMS, can be documented as well as any other, perhaps non-computerized systems, such as organization structures and business plans. For example, data shared between DB2 and IMS may be documented once only and maintained centrally.
- Historical, production and development versions of a system and its components may be documented
- Populated using the following methods:
  - Using diagrams created on your Intelligent Workstation
  - Automatically, as the result of using the design tool
  - Directly, using definition statements
  - Importing information.
- Interrogation and reporting capabilities.

Refer to the *ASG-ControlManager User's Guide* for details of using a dictionary.

## **Design Diagramming Tool**

The following list highlights the features of the Diagramming Tool:

- Easy drawing of structured design diagrams including, for example, Entity Models and organizational charts on your Intelligent Workstation
- Diagrams can be loaded to a dictionary on the host (mainframe) system
- Diagram validation for consistency and logic according to predefined rules
- A local dictionary to which you can download a subset of the host dictionary
- Information in the host dictionary can be locked while you have it on the Intelligent Workstation
- The host dictionary can be interrogated while diagrams are being created
- Diagram objects, connectors, menus, and validation rules can all be tailored to suit your environment.

Refer to the *ASG-ManagerView User's Guide* for details of the diagramming tool.

## **Database Design Tool**

The following list highlights the features of the Design Tool:

- 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 dictionary with a first-cut physical design made up of tables, views, and indexes.

## **Data Definition Language (DDL) Generator**

The following list highlights the features of the DDL Generator:

- Can generate DDL statements for the major DBMS such as DB2, SQL/DS and IMS
- Generates SQL statements (CREATE, ALTER, DROP, GRANT, etc.) from dictionary 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, "Implementation and Maintenance," on page 95 for details of DDL generation for SQL/DS.

### **COBOL, PL/I, and ASSEMBLER Generator**

The following list highlights the features of the source language generator:

- Generates host language data structures for inclusion in programs which 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 installation's conventions and standards
- Can generate table layouts documenting the columns, edit procedures, field procedures, and validation procedures of tables and views.

Refer to Chapter 5, "Implementation and Maintenance," on page 95 for details of source language generation for SQL/DS.

### **Dynamically Submitting SQL Statements to DB2 or SQL/DS**

The following list highlights the features of Dynamic SQL Services:

- 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 which 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 Manager Products Procedures Language variables.

Refer to Chapter 7, "Dynamic SQL Services," on page 107 for details of Dynamic SQL Services.

## Importing Information from SQL/DS

The following list highlights 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 dictionary 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 dictionary standards.
- Generates member definition statements for the proposed members in layouts which may be tailored to suit your dictionary standards
- Enters the proposed members into the dictionary.

Refer to Chapter 8, "Import," on page 123 for details of importing information.

## Functions: How to Use the Tools We Provide

### Introduction

Each of the features we provide supports part of a DB2 or SQL/DS system life cycle. The life cycle consists of the following phases:

- Design
- Implementation
- Maintenance.

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

### Standards

Our tools enable you to implement and control standards throughout the life cycle. They are particularly useful in the context of:

- Naming standards
- Database development standards
- Application development standards.

If your systems are documented in the dictionary, 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 will be an easy matter if you have naming standards.

You can record several alternative names (aliases) for each definition in the dictionary, to suit different application environments. When you come to generate SQL statements, COBOL, PL/I, and/or ASSEMBLER source direct from the dictionary, you can use the relevant alias name instead of the unique name by 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.

So, you can help your development teams to adhere to your standards by providing, via the dictionary, 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 dictionary.

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 dictionary or in the user definable HELP system that we provide will ensure that they are centrally available.

So, if you control your systems life-cycle using our tools, you can help ensure that the systems you build conform to your standards. The result will be systems which are easier to understand and more effective communications.

## **SQL/DS Database Design**

### **Overview**

DB2 and SQL/DS database design involves 2 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.

And, with DB2 and SQL/DS, you need to take Referential Integrity into account in both stages.

### **Identifying Data and Functional Requirements**

To begin with you must identify the data to be used in the database and define it in the form of:

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

**Userview Models.** Definitions of the databases' end-users' requirements.

You can use the Diagram Editor to draw Entity Models on an Intelligent Workstation and then upload them to the host dictionary where they are converted into and held as definitions. Definitions in the dictionary can be interrogated, reported, and used by the design tool. The Diagram Editor will check your diagrams, according to predefined rules, for validity, consistency, and completeness.

The host dictionary is always available while you are using the Diagramming Tool. You can interrogate and report from the dictionary, to obtain the information you need, while you are creating your entity relationship diagrams. If you are updating information which is already in the dictionary, 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 dictionary.

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

If you have used our design tool to design other DBMS databases and that work was based on an Entity Model that is still current, you can re-use that Entity Model and associated data element documentation already in your dictionary, 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 from it, 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.

Refer to Chapter 3, "SQL/DS Database Design," on page 25 for details of the SQL/DS design process.

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

### Logical Database Design

Once the Entity Models and Userviews are documented in the dictionary, 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 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 refine the design by adjusting your Entity and Userview Models and iterating design on the WBDA until you are satisfied with the design and you understand its referential structures.

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

### **Physical Database Design**

Once you are satisfied with the relational schema, it can be converted, automatically, 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 dictionary can be populated, automatically, 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 dictionary.

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.

A SIZE function is also provided to enable you to estimate the storage space required by tables.

You can use the dictionary 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 dictionary to check which tables are stored in which DB2 storage groups.

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

You complete the physical design as follows:

- Document the other object types required: SQL/DS dbspaces or DB2 databases, storage groups and table spaces as well as additional (non-primary) DB2 and SQL/DS 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 dictionary 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 dictionary 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 4, "Dictionary Definition," on page 81 for details of SQL/DS dictionary definitions.

## **Implementation**

### **Implementing Your Design**

When your database is fully documented in the dictionary, you can generate, automatically, 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 Services enable you to submit SQL statements to DB2 or SQL/DS from within the Manager Products environment.

Refer to Chapter 5, "Implementation and Maintenance," on page 95 for details of SQL generation for SQL/DS.

### **Developing Applications**

Having implemented the design you will already have started work on building application programs which 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 dictionary. (Data declaration statements define the DB2 tables and views that the application accesses.)

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 application programs.

Implementation will be smoother if, as discussed earlier in this branch, you have used the dictionary to help you impose naming standards.

When you come to generate COBOL, PL/I, and/or ASSEMBLER 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 dictionary, the impact of changes to tables and views etc. can be measured easily: by interrogating the dictionary. Thus the dictionary becomes an intelligent and active part of your change-control procedures.

Refer to Chapter 5, "Implementation and Maintenance," on page 95 for details of COBOL, PL/I, and ASSEMBLER generation for SQL/DS.

### ***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 dictionary, and then
- Regenerating the necessary components from the dictionary.

For example, unnormalizing certain tables for performance reasons may involve combining tables.

You can:

- Interrogate the dictionary 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

### Overview

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 dictionary
- Interrogation. For example: 'where else is this field used?'
- Reporting. You want, for example, 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 (adding a foreign key, for example)
- 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 Control in the System Life Cycle

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

Use the status facility to maintain both production and development versions of the documentation in the dictionary.

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 for a new/changed version of the system under development, without affecting the production system and without being isolated from it.

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

### Interrogation and Reporting

You can:

- Interrogate
- Report from
- Produce documentation of table layouts from

the dictionary 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.

You can index dictionary definitions using keywords/classifications which 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.

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

Refer to "SQL-TABLE" on page 245 for details of SQL/DS table layouts.

Refer to "Dynamic SQL Services" on page 13 for details of Dynamic SQL Services.

### **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 dictionary you get a complete impact analysis report of what you will lose or affect before you drop the object.

**Note:** \_\_\_\_\_

Although DB2 will not let you drop a storage group containing table spaces, it may be useful to use this command to see what is in the storage group.

---

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 dictionary recreation or the object is very simply done.

You can also generate ALT TABLE statements from the definitions of tables in the dictionary. 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 generate GRANT and REVOKE statements, automatically, from the dictionary.

This is particularly useful if you have dropped a DB2 database or SQL/DS dbspace (and everything that's inside it) because you will, usually, need to re-GRANT many authorizations when the objects are re-created.

### **Application Maintenance**

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

You can obtain the information you need to make such decisions from the dictionary. 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 dictionary definition and regenerate data structures and DB2 data declaration statements easily.

### **Dynamic SQL Services**

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

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

You can submit SQL statements which have been previously generated from the dictionary 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 Services.

Refer to Chapter 7, "Dynamic SQL Services," on page 107 for details of Dynamic SQL Services.

### **Import Facilities**

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 Dictionary/Repository. The major benefits of the import facilities are that:

- Manager Products users who have not documented their DB2 or SQL/DS environment in the dictionary 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 dictionary you can use Manager Products' CASE tools to analyze, maintain, and improve that environment.

Powerful dictionary 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.

The Design Diagramming and Database Design tools enable you to analyze the dictionary in order to produce a normalized logical design from which the dictionary can be populated with a first-cut physical design. The import facilities therefore provide the first step in the reengineering process.

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

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

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

## **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 dictionary.
- The physical design is completed and documented in the dictionary 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 dictionary in order to obtain information about the system
  - Generating ALTER statements from the dictionary to reflect changing requirements for tables
  - Supporting security and authorizations in the database.

## **Benefits**

### **Introduction**

The corporate Dictionary/Repository is a central repository of re-usable 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 Dictionary/Repository and the productivity tools based upon it, are described in this section.

### ***A Shared and Re-usable 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 Dictionary/Repository you can create a central data model in which each data element used by the corporation is defined:

- Once only,
- In a consistent manner,
- Independent of any particular DBMS/application.

Using the import facilities we provide, you can populate the Corporate Dictionary/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 dictionary 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 (branches and departments, for example).

So, the Corporate Dictionary/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 Dictionary/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, and Databases
  - Developing systems
  - Organizational units (departments, branches, etc.).

- A data model which 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 dictionary helps developers to find out what data already exists
- Data sharing 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 dictionary 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 re-use.

All of the above reduce the:

- Time,
- Resource,
- Confusion, and
- Expense

involved in building the information systems required to satisfy your organizations 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 Dictionary/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, and
- Referential Integrity

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

Using our design tool the database designer is free to devote himself/herself to aspects of the design process that require human judgment:

- 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.

Again, 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 Dictionary/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 Intelligent 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 PC 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?

---

### Introduction

The purpose of this chapter is to direct users who are unfamiliar with Manager Products to the documentation relevant to the task they wish to perform. For each task the relevant section and page reference in this manual is given.

### ASG Support for Your SQL/DS Environment

| <b>Topic</b>                               | <b>Page</b> |
|--|-------------|
| SQL/DS Database Design                     | 19          |
| SQL/DS Dictionary Definition               | 20          |
| SQL/DS Object Generation                   | 21          |
| Dynamically Submitting SQL Statements      | 22          |
| Importing Information about SQL/DS Objects | 22          |

### SQL/DS Database Design

#### *Producing Output Describing the SQL Design*

| <b>Topic</b>                    | <b>Page</b> |
|---------------------------------|-------------|
| SQL LIST TABLES                 | 198         |
| SQL LIST CYCLES                 | 197         |
| SQL REPORT                      | 240         |
| SQL PLOT CLUSTER                | 200         |
| SQL PLOT REFERENTIAL-STRUCTURES | 203         |

## **Generating and Populating SQL Dictionary Definitions of Specified Member Type**

| <b>Topic</b>                                | <b>Page</b> |
|---|-------------|
| Generating and Populating SQL-TABLE Members | 20          |
| Generating and Populating SQL-INDEX Members | 211         |
| Generating and Populating SQL-VIEW Members  | 212         |
| Combining SQL POPULATE Command Options      | 216         |

### **Generating and Populating SQL-TABLE Members**

| <b>Topic</b>                                  | <b>Page</b> |
|---|-------------|
| Generating and Populating SQL-TABLE Members   | 209         |
| Suppressing Support for Referential Integrity | 210         |
| Generating References to Dbspaces             | 210         |

## **SQL/DS Dictionary Definition**

### **Documenting an SQL/DS Dictionary Schema**

| <b>Topic</b>                            | <b>Page</b> |
|---|-------------|
| SQL/DS Object Definition                | 20          |
| Documenting SQL/DS Security Information | 88          |
| Naming Conventions for SQL/DS Members   | 89          |
| Processing Your SQL/DS Members          | 92          |

### **SQL/DS Object Definition**

| <b>Topic</b>                                       | <b>Page</b> |
|--|-------------|
| Documenting the Columns of SQL/DS Tables and Views | 84          |
| SQL-DBSPACE  | 184         |
| SQL-INDEX  | 192         |
| SQL-PRIVILEGE                                      | 229         |
| SQL-TABLE  | 245         |
| SQL-USER   | 259         |
| SQL-VIEW   | 262         |

## SQL/DS Object Generation

### *Generating SQL Statements and SQL/DS Host Language Data Structures*

| <b>Topic</b>  | <b>Page</b> |
|---|-------------|
| SQL ACQUIRE   | 172         |
| SQL CREATE  | 182         |
| SQL SYNONYM   | 244         |
| SQL DROP  | 187         |
| SQL ALTER   | 173         |
| SQL COMMENT and SQL LABEL   | 180         |
| SQL GRANT and SQL REVOKE  | 190         |
| SQL PRODUCE   | 236         |
| Documenting the Columns of SQL/DS Tables and Views                          | 84          |
| SQL SIZE  | 243         |
| Defining an AS Clause   | 277         |
| Submitting Generated Output to Your Relational Environment                  | 97          |
| Generating Tailored SQL Statements and SQL/DS Host Language Data Structures | 21          |

### *Generating Tailored SQL Statements and SQL/DS Host Language Data Structures*

| <b>Topic</b>  | <b>Page</b> |
|---|-------------|
| Displaying Internal Diagnostic Output   | 100         |
| Generating Object Names and External Names from Aliases                                     | 101         |
| Generating SQL CREATE, LABEL ON, and COMMENT ON Statements from One Member at the Same Time | 102         |
| Generating a Host Language Indicator Structure  | 103         |
| Tailoring DATE and TIME Character Field Lengths   | 104         |

## **Dynamically Submitting SQL Statements**

| <b>Topic</b>  | <b>Page</b> |
|---|-------------|
| ISQL  | 149         |
| Creating and Populating a Table                             | 115         |
| Inserting Rows into a Table                                 | 117         |
| Importing Information and Assigning it to Command Variables | 118         |
| Submitting any SQL Statement That Can be Prepared           | 120         |
| Creating Your Own HELP Text                                 | 122         |

## **Importing Information about SQL/DS Objects**

| <b>Topic</b>              | <b>Page</b> |
|---------------------------|-------------|
| EXTRACT SQL               | 143         |
| RECONCILE                 | 157         |
| PREVIEW                   | 153         |
| POPULATE                  | 151         |
| Tailoring Import Commands | 128         |

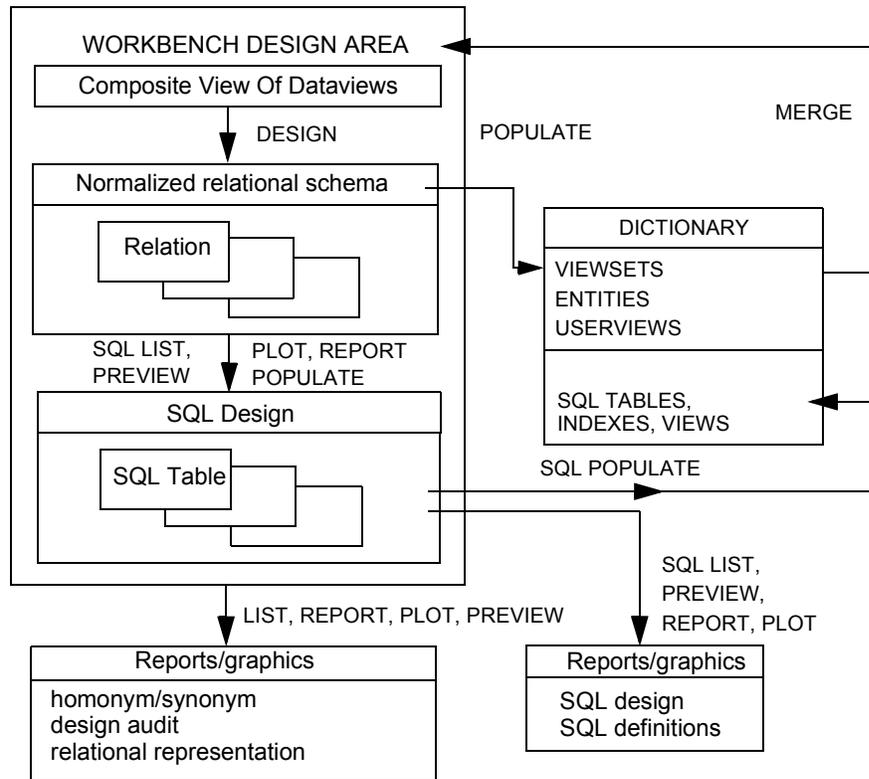


Figure 1 Overview Diagram of SQL/DS Database Design

Reading from top to bottom, this diagram represents the support provided for SQL/DS at the different stages of the SQL/DS database design process. 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 SQL commands are also shown.



---

# 3

## SQL/DS Database Design

---

### Introduction to SQL/DS Database Design

#### Overview

There are two major stages of SQL/DS 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.

SQL/DS 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 SQL/DS 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 dbspaces containing referential structures.

You can automate much of the SQL/DS 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 Workbench Design Area (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 SQL design in the WBDA from the relational schema, using any of the SQL LIST, SQL PLOT, SQL REPORT, SQL PREVIEW, or SQL POPULATE commands. The SQL design consists of SQL tables, with their primary and foreign keys, as well as table indexes and views.
- Evaluate the tables in the SQL design, using output from the SQL LIST, SQL PLOT, and SQL REPORT commands

- Use the output from the SQL LIST CYCLES and PLOT REFERENTIAL-STRUCTURES commands to help plan which dbspaces 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 SQL-TABLE, SQL-INDEX, and/or SQL-VIEW members, using the SQL 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 SQL POPULATE command
- Complete the physical design of the SQL/DS 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 SQL command after you have generated the relational schema. However, referential integrity is not supported by this command.

---

Refer to "Designing a SQL/DS Database" on page 29 for a detailed description of the SQL/DS database design process.

## **Support for Referential Integrity**

SQL/DS 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.

SQL/DS uses the primary keys and foreign keys in SQL tables to enforce RI.

ASG-DesignManager supports RI throughout the SQL/DS 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 dbspaces containing referential structures.

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

When you use the SQL PREVIEW or the SQL POPULATE command to generate SQL-TABLE dictionary definitions from the tables in the Workbench Design Area (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 SQL PLOT and SQL 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 dbspaces in which the SQL tables should be stored.

The report outputs can help you design the referential constraints for the delete rules which apply to the SQL 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 dbspaces in which the tables are to be stored.

The output of the SQL PLOT REFERENTIAL-STRUCTURES command displays the referential structures present in the SQL design generated in the Workbench Design Area. The SQL 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 SQL-TABLE member by updating the FOREIGN-KEY clause.

Refer to "SQL LIST CYCLES" on page 197 for details of the SQL LIST CYCLES command.

Refer to "SQL PLOT REFERENTIAL-STRUCTURES" on page 203 for details of the SQL PLOT REFERENTIAL-STRUCTURES command.

## **Features to Support SQL/DS**

The relational schema in the Workbench Design Area (WBDA) is automatically converted to a SQL design containing SQL tables, indexes, and views when any of the following commands is entered:

- SQL LIST
- SQL REPORT
- SQL PLOT
- SQL PREVIEW
- SQL POPULATE.

SQL referential integrity is fully supported.

The SQL LIST TABLES command produces a list of some or all of the tables in the SQL design generated in the WBDA.

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

The SQL 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.

The SQL 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.

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

Once the SQL design has been generated, you can generate dictionary definitions for SQL tables, indexes, and views. The SQL PREVIEW command enables you to preview the generated SQL dictionary definitions before populating the dictionary with them.

The SQL POPULATE command automatically populates the dictionary with the following member types (if you have the optional SQL/DS Definition facility installed):

- SQL-TABLE
- SQL-INDEX
- SQL-VIEW
- SYSTEM.

You can also use the PRODUCE SQL command, if you want to generate SQL CREATE TABLE statements directly from the relations in the relational schema.

## Designing a SQL/DS 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 SQL/DS database design process by defining your data in the dictionary. The definitions are in the form of:

**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 and

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

Entities and userviews are collectively known as data-views.

You should be careful to avoid homonyms and synonyms when naming your data elements. USERVIEW and ENTITY members should refer to validly named data elements.

### Generating a Relational Schema

After you have created your entity and userview models in the dictionary, you can create a relational schema. The process of creating a relational schema is described below.

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

Next, you 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 SQL 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 SQL design from it in the WBDA by issuing any ASG-DesignManager SQL 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.

If you want to, you can use the PRODUCE SQL command at this point to generate SQL CREATE TABLE statements straight from the WBDA.

## **Generating the SQL Design**

The SQL design comprises tables generated directly from the relations in the relational schema in the Workbench Design Area (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 SQL LIST, SQL PLOT, or SQL REPORT command to generate and report the SQL 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 SQL design by entering the SQL PREVIEW or SQL POPULATE commands, but these commands do not provide a detailed report of the tables in the SQL design.

The following table shows the correspondence between the relations in the relational schema and the tables in the SQL design.

| <b>Relational Schema</b>   | <b>SQL Design</b>  |
|--|--|
| 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 of 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 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 (except in the case of a domain association; see below)   |
| Foreign key, hierarchical-one, and domain associations   | Foreign key relationships  |

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

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

### **Reporting the SQL Design**

Use the SQL LIST, SQL PLOT, and SQL REPORT commands to produce listings, plots, and reports of the tables in the SQL design.

Any of these commands will cause the relational schema to be converted to a SQL 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 SQL 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 SQL 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 dbspaces 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 SQL design.

### **Populating the Dictionary with SQL Members**

When you are satisfied with the SQL tables in the Workbench Design Area (WBDA), you can begin generating SQL-TABLE, SQL-INDEX, and SQL-VIEW dictionary member definitions.

You can also generate a SYSTEM dictionary definition containing a list of the names of all the SQL 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 SQL PREVIEW command. This generates and reports SQL dictionary definitions from the SQL 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 SQL 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 SQL POPULATE command to populate the dictionary with them. SQL PREVIEW and SQL 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 SQL 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 SQL-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 dbspaces at this stage. You may choose to have one dspace per table, or to store a whole referential structure in one dspace, or to have one or more referential structures spanning dbspaces.

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 SQL PLOT REFERENTIAL-STRUCTURES and SQL LIST CYCLES commands.

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

In addition, a SQL-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 SQL 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 "Introduction to Referential Structures and Cycles" on page 27 for a discussion of referential structures and cycles.

## Examples of the SQL/DS Database Design Process

### Introduction to Examples

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

The two examples are called:

- The Department Model
- The Parts Model.

In both examples, when displaying the relational schema the convention followed is that associations are directed from the source relation (containing the foreign key) to the target relation (containing the corresponding primary key); when displaying the SQL design the convention followed is that relationships are directed from the parent table to the dependent table, that is, in 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.

#### Department Model Example: The Entity Model

This example describes three entities in an organization; they represent employees, departments, and offices, and their interrelationships, as shown in the following diagram.

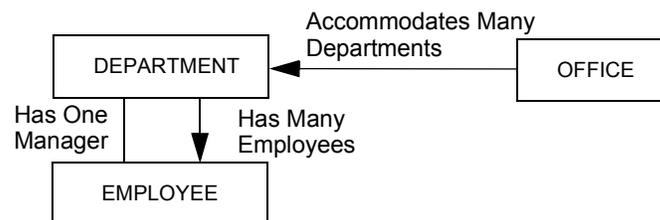


Figure 2 Department Model: Entities and their Interrelationships

You begin by defining the ENTITY members in the dictionary, and calling them EMPLOYEE-ENT, MANAGER-ENT, DEPARTMENT-ENT, and OFFICE-ENT, as follows:

EMPLOYEE-ENT

ENTITY

IDENTIFIER IS EMPLOYEE-NO  
ONE-ATTRIBUTES ARE EMPLOYEE-NAME  
SUB-ENTITIES ARE MANAGER-ENT

MANAGER-ENT

ENTITY

IDENTIFIER IS MANAGER-NO

DEPARTMENT-ENT

ENTITY

IDENTIFIER IS DEPARTMENT-NO  
ONE-ATTRIBUTES ARE DEPARTMENT-NAME  
ONE-ASSOCIATION TO MANAGER-ENT  
MULTI-ASSOCIATION TO EMPLOYEE-ENT

OFFICE-ENT

ENTITY

IDENTIFIER IS OFFICE-LOCATION  
ONE-ATTRIBUTES ARE OFFICE-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.

### Department Model Example: The Composite View

If the entities EMPLOYEE-ENT, DEPARTMENT-ENT, and OFFICE-ENT are merged into the Workbench Design Area (WBDA), using the MERGE command, the following dependencies are derived.

**Table 1 Department Model: Derived Dependencies**

| Dependency      |      | Derived from Entity               |
|-----------------|------|-----------------------------------|
| EMPLOYEE-NO     | →    | EMPLOYEE-NAME<br>EMPLOYEE-ENT     |
| MANAGER-NO      | = => | EMPLOYEE-NO<br>EMPLOYEE-ENT       |
| DEPARTMENT-NO   | →    | DEPARTMENT-NAME<br>DEPARTMENT-ENT |
| DEPARTMENT-NO   | →    | MANAGER-NO<br>DEPARTMENT-ENT      |
| DEPARTMENT-NO   | →    | EMPLOYEE-NO<br>DEPARTMENT-ENT     |
| OFFICE-LOCATION | →    | OFFICE-NAME<br>OFFICE-ENT         |
| OFFICE-LOCATION | →    | DEPARTMENT-NO<br>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.

### Department Model Example: Contents of the Relations

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

The following table 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 2 Department Model: Relations and their 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   |            |

**Department Model Example: Associations Between the Relations**

The diagram below 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.

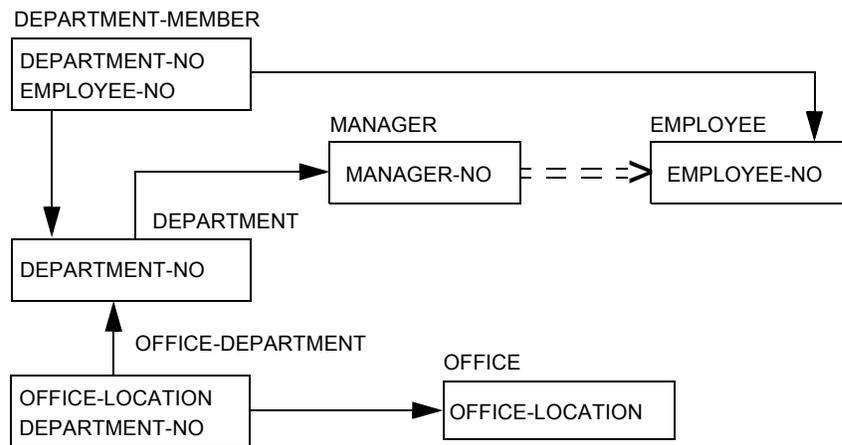


Figure 3 Department Model: Associations of the Relations

**Department Model Example: Description of 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
- A 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
- 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
- A non-prime data element, employee-name
- No associations leading from it.

OFFICE is an FD relation. It has:

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

### Department Model Example: Contents of the Tables

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

- SQL LIST
- SQL REPORT
- SQL PLOT
- SQL PREVIEW
- SQL POPULATE.

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

When a table in the SQL design is generated from a relation in the relational schema, the table takes the name and Workbench Design Area 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 SQL 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 3 Department Model: Contents of the Tables**

| <b>Table Name</b> | <b>Columns Comprising Table</b> |                 |            |
|-------------------|---------------------------------|-----------------|------------|
| 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   |            |

### Department Model Example: Relationships Between the Tables

The diagram below represents the generated SQL design. It displays all six tables with their respective primary key columns and the foreign key relationships in which they participate.



DEPARTMENT-MEMBER is a dependent table. It has:

- A composite primary key with DEPARTMENT-NO and EMPLOYEE-NO as the component prime columns
- A parent table DEPARTMENT. In this relationship, the prime column, DEPARTMENT-NO, of DEPARTMENT-MEMBER is the foreign key to DEPARTMENT.
- A 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
- A parent table DEPARTMENT. In this relationship, the prime column, DEPARTMENT-NO, of OFFICE-DEPARTMENT is the foreign key to DEPARTMENT.
- A 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
- A non-prime column, office-name
- A dependent table OFFICE-DEPARTMENT in which the primary key component, OFFICE-LOCATION, is the foreign key to OFFICE.

### **Department Model Example: Refining the SQL Design**

So far the example has illustrated how a SQL design is generated in the Workbench Design Area (WBDA). When you enter the SQL PREVIEW or SQL POPULATE command, dictionary definitions are generated automatically for the six SQL 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 the following diagram:

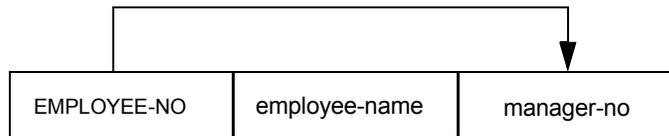


Figure 5 Department Model: A Self-referencing Table

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 of the SQL/DS database by documenting those decisions in the relevant SQL-TABLE, SQL-INDEX, and SQL-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.

### Parts Model Example: The Entity Model

In this example, there is an entity called PART. Its relationship with other manufacturing parts is represented as:

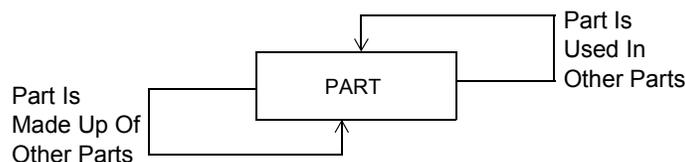


Figure 6 Parts Model: Interrelationships of Parts

You begin by defining two ENTITY members in the dictionary, and calling them PART-ENT and COMPONENT-ENT. You should define the subentities MAJOR-PART-ENT and MINOR-PART-ENT as subentities of PART-ENT.

```
PART-ENT
ENTITY
IDENTIFIER IS          PART-NO
ONE-ATTRIBUTES ARE    PART-NAME, PART-PRICE, PART-QTY- IN-STOCK
SUB-ENTITIES ARE      MAJOR-PART-ENT, MINOR-PART-ENT

MAJOR-PART-ENT
ENTITY
IDENTIFIER IS          MAJOR-PART-NO

MINOR-PART-ENT
ENTITY
IDENTIFIER IS          MINOR-PART-NO

COMPONENT-ENT
ENTITY
IDENTIFIER IS          MAJOR-PART-NO, MINOR-PART-NO
ONE-ATTRIBUTES ARE    ASSEMBLY 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
- 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:

- A composite identifier consisting of the attributes, MAJOR-PART-NO and MINOR-PART-NO
- A one-attribute, assembly-quantity, specifying the number of each minor part used in the assembly of the major part.

### Parts Model Example: The Composite View

When the entities PART-ENT and COMPONENT-ENT are merged into the Workbench Design Area (WBDA) using the MERGE command, the following dependencies are derived.

**Table 4 Parts Model: 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 ASG-DesignManager derives dependencies from ENTITY definitions.

### Parts Model Example: Contents of the Relations

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 Workbench Design Area.

The following table 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 5 Parts Model: Relations and their 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                     | part-no       |                   |                   |
| COMPONENT  | MAJOR-PART-NO                     | MINOR-PART-NO | assembly-quantity |                   |

### Parts Model Example: Associations between the Relations

The diagram below 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.

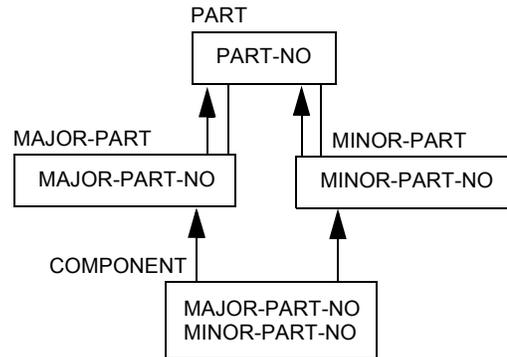


Figure 7 Parts Model: Associations between the Relations

### Parts Model Example: Description 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
- 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
- A 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
- A 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.

### Parts Model Example: Contents of the Tables

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

- SQL LIST
- SQL REPORT
- SQL PLOT
- SQL PREVIEW
- SQL POPULATE.

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

When a table in the SQL design is generated from a relation in the relational schema, the table takes the name and Workbench Design Area 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 SQL 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 6 Parts Model: Contents of the Tables**

| 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 |                   |

### Parts Model Example: Relationships between the Tables

The diagram below represents the generated SQL design. It displays all four tables with their respective primary key columns and the foreign key relationships in which they participate.

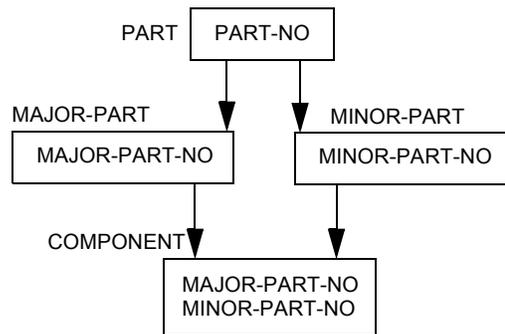


Figure 8 Parts Model: Relationships of the Tables

### Parts Model Example: Description of the Tables

PART is a parent table. It has:

- The primary key, PART-NO
- Non-prime columns, part-name, part-price, and part-qty-in-stock
- A dependent table MAJOR-PART in which the primary key, MAJOR-PART-NO, is a foreign key to PART
- A 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
- A 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
- A 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
- A non-prime column, assembly-quantity
- A parent table, MAJOR-PART. In this relationship, the prime column, MAJOR-PART-NO, of COMPONENT is the foreign key to MAJOR-PART.
- A parent table, MINOR-PART. In this relationship, the prime column, MINOR-PART-NO, of COMPONENT is the foreign key to MINOR-PART.

### Parts Model Example: Refining the SQL Design

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

Once the SQL 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 follows:

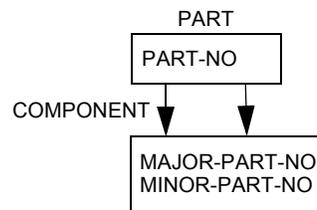


Figure 9 Parts Model: Merging Tables

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 SQL 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 SQL/DS database by documenting those decisions in the relevant SQL-TABLE, SQL-INDEX, and SQL-VIEW dictionary members.

## SQL/DS Design Analysis

### Output from the SQL REPORT Command

#### Introduction to the SQL REPORT Output

The SQL REPORT command produces a report, called the SQL Table Report, showing the SQL tables generated in the SQL design. You can report all the tables in the Workbench Design Area (WBDA) or a selection of tables. Tables may be selected by name or WBDA number.

The output includes first the total number of tables in the Workbench Design Area (WBDA). It then gives details of the contents of each selected table, a description of each dependency represented by the table, and the origin of that dependency, and, for each foreign key relationship in which the table participates, information about the related parent or dependent table.

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

Refer to *ASG-DesignManager User Formatted Output* for details of the User Formatted Output facility.

#### Contents of Tables in SQLREPORT Output

For each selected table, the SQL 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.

Then 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 type of dependency: functional (FD), multivalued (MVD), or domain (DD).

The second output line shows the origin of the dependency:

- Either it was generated by the MERGE command from one or more data-views, in which case 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 "Example of SQL REPORT Output" on page 50 for an example of the SQL REPORT output.

### **Foreign Key Relationships in SQL REPORT Output**

For each selected table in the SQL design which participates in foreign key relationships, the SQL 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 Workbench Design Area (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 SQL 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, that is, 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 SQL 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 director 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 of SQL REPORT Output***

In this example, a SQL Table Report is produced for the following tables which have been generated in the SQL design in the Workbench Design Area (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 7 Tables in the SQL Design in the WBDA: SQL Report**

| 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 SQL Table Report are described in order of table number, as they appear in the list above.

```

*****
*
*          SQL TABLE REPORT          *
*
*          TOTAL NUMBER OF TABLES   . . . 6
*
*****

```

```

=====LEAF/DEPENDENT=====
1 DEPARTMENT-MEMBER
=====
PRIMARY KEY          DATA ELEMENTS
DEPARTMENT-NO
EMPLOYEE-NO

```

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



```

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

| ABS | LEFT-HAND-SIDE | TYPE | RIGHT-HAND-SIDE |
|-----|----------------|------|-----------------|
| REL | DATA-VIEW      |      |                 |
| 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<br>EMPLOYEE-NO     |                           |
| F-KEY           | DEPARTMENT-NO                    | DEPARTMENT-NO             |
| NAME            | 2 OFFICE-DEPARTMENT              |                           |
| TYPE            | FOREIGN-KEY                      |                           |
| P-KEY           | OFFICE-LOCATION<br>DEPARTMENT-NO |                           |
| F-KEY           | DEPARTMENT-NO                    | DEPARTMENT-NO             |

```

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

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

| Parent Table |                              | Current Table Primary Key |
|--------------|------------------------------|---------------------------|
| NAME         | 1 DEPARTMENT-MEMBER          |                           |
| TYPE         | FOREIGN-KEY                  |                           |
| P-KEY        | DEPARTMENT-NO<br>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 KEY          DATA ELEMENTS
OFFICE-LOCATION      OFFICE-NAME
  
```

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

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

```

=====PARENT/DEPENDENT=====
6  MANAGER
=====
PRIMARY KEY          DATA ELEMENTS
MANAGER-NO
    
```

| ABS | LEFT-HAND-SIDE | TYPE    | RIGHT-HAND-SIDE |
|-----|----------------|---------|-----------------|
| REL | DATA-VIEW      |         |                 |
| 2   | MANAGER-NO     | ==DD==> | EMPLOYEE-NO     |
| 2   | ENTITY         |         | 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 SQL PLOT CLUSTER Command

### Introduction to the SQL PLOT CLUSTER Output

The output of the SQL PLOT CLUSTER command, called the SQL Cluster Plot, provides you with detailed graphical displays for selected tables in the SQL design in the Workbench Design Area (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 SQL 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 SQL 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 SQL 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 SQL Cluster Plot.

Refer to *ASG-DesignManager User Formatted Output* for details of the User Formatted Output facility.

### Format of the Cluster Diagram

In each cluster diagram displayed in the SQL 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 SQL/DS 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 Workbench Design Area (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.

Refer to "Foreign Key Relationships in SQL REPORT Output" on page 49 for a discussion of direct and domain types of foreign key relationships.

### ***The Information in the Cluster Diagram***

Each cluster diagram of the SQL 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 Workbench Design Area (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).

**Example of the Output from SQL PLOT CLUSTER**

Consider the following example. In the table below 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.

**Table 8 Tables in the SQL DESIGN in the WBDA: SQL PLOT**

| 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                               |

Taking each of these tables in turn, the following clusters would be displayed in the SQL Cluster Plot.

When the table DEPARTMENT-MEMBER is selected:

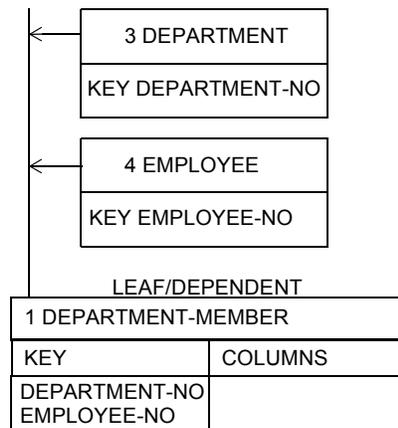


Figure 10 Cluster Generated for the Table DEPARTMENT-MEMBER

When the table OFFICE-DEPARTMENT is selected:

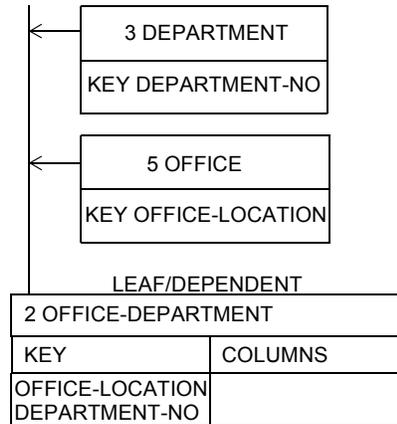


Figure 11 Cluster Generated for the Table OFFICE-DEPARTMENT

When the table DEPARTMENT is selected:

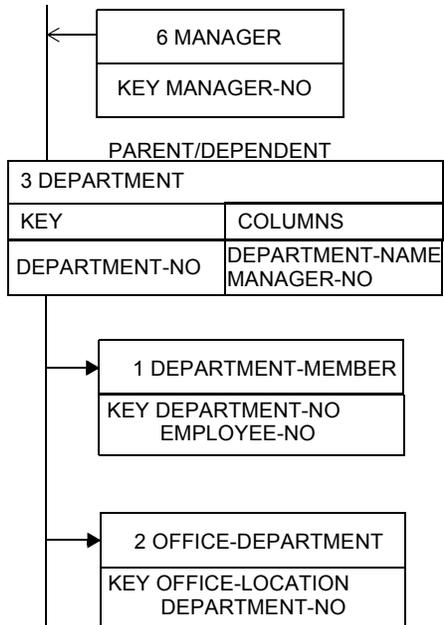


Figure 12 Cluster Generated for the Table DEPARTMENT

When the table EMPLOYEE is selected:

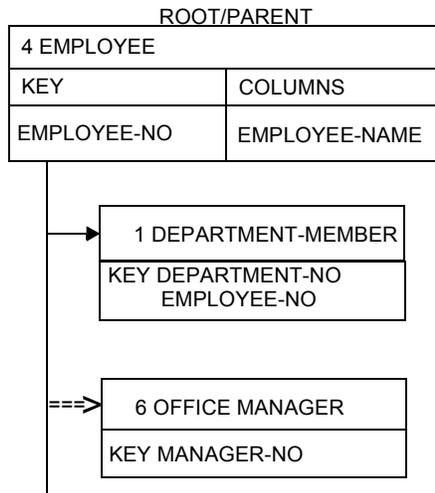


Figure 13 Cluster Generated for the Table EMPLOYEE

When the table OFFICE is selected:

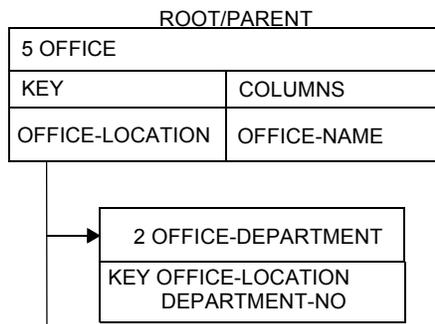


Figure 14 Cluster Generated for the Table OFFICE

When the table MANAGER is selected:

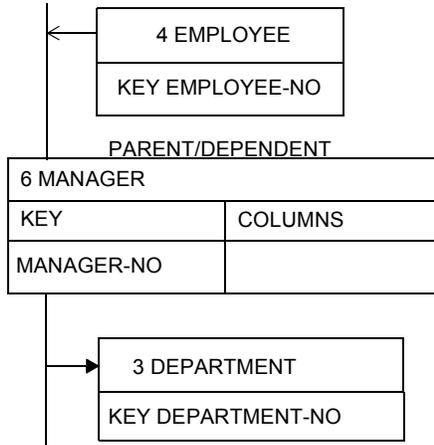


Figure 15 Cluster Generated for the Table MANAGER

### SQL Design Relationship Matrix

In the SQL Cluster Plot, the SQL 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 Workbench Design Area (WBDA), regardless of any selections specified in the SQL 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 here is the one produced after the cluster diagrams discussed earlier in this branch.

|                   |   | 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 |   |   |   |

Figure 16 Matrix Produced by the SQL PLOT Command

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 of 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 "Example of the Output from SQL PLOT CLUSTER" on page 59 to see the associated cluster plot.

## Output from the SQL PLOT REFERENTIAL-STRUCTURES Command

### Introduction and Overview to the SQL PLOT REFERENTIAL-STRUCTURES Output

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

The SQL Referential Structures Plot is complementary to the SQL Cluster Plot, which, for each SQL table in the SQL design, provides details of its content and its relationships with other SQL tables. The SQL 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 SQL Referential Structures Plot include the following types:

- Direct foreign key relationships
- Domain relationships.

Refer to "Foreign Key Relationships in SQL REPORT Output" on page 49 for details of these relationship types and how they correspond to association types in the relational schema from which the SQL design was generated.

The tables displayed in the SQL Referential Structures Plot can be of any of the following types:

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

Refer to "Description of the SQL LIST TABLES Output" on page 73 for definitions of the various types of tables.

In a SQL 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 SQL design, you would find that the number of arrows that cross one another would, in general, create a mass of confusing detail.

In the SQL 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) SQL 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 SQL 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 children's children, and so on, are called its *descendants*.

In addition to the representation of the SQL design as an equivalent hierarchical structure, a further simplifying feature has been incorporated in the SQL 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. Each 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.

Refer to "Use of Pointers in the SQL PLOT REFERENTIAL-STRUCTURES Output" on page 67 for details of the use of pointers.

In a SQL 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 SQL Referential Structures Plot which can further simplify the display. You can show either:

- All the tables of the SQL 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 SQL PLOT command.

Refer to *ASG-DesignManager User Formatted Output* for details of the User Formatted Output facility.

### ***Layout of the SQL PLOT REFERENTIAL STRUCTURES Output***

In summary, in the SQL Referential Structures Plot, the tables in the SQL design are represented by boxes and pointers; relationships between the tables are represented by connecting lines. In the 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 eleven 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.

Following are examples of text formed from table keys:

| <b>Key</b>   | <b>Text Formed</b>                       |  |
|--|--|--|
| EMPLOYEE-NO  | EMPLOYEE-NO                              | (no truncation)  |
| EMPLOYEE-NAME  | EMPLOYEE-NA                              | (1st 11 characters)  |
| DEPARTMENT-NO,<br>EMPLOYEE-NAME,<br>OFFICE-NO                | DEPARTMENT-<br>EMPLOYEE-NA<br>OFFICE-NO  | (1st 11 characters)<br>(1st 11 characters)<br>(no truncation)              |
| DEPARTMENT-NO,<br>EMPLOYEE-NAME,<br>OFFICE-NO,<br>PROJECT-NO | DEPARTMENT-<br>EMPLOYEE-NA<br>OFFICE-NO, | (1st 11 characters)<br>(1st 11 characters)<br>(no truncation)<br>(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 SQL Referential Structures Plot appears in the following diagram:

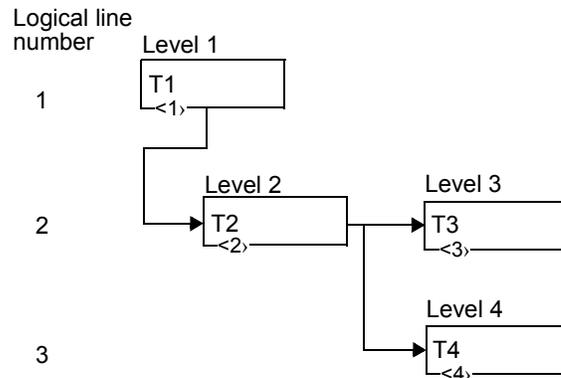


Figure 17 SQL Referential Structure Plot

In the diagram, 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 SQL PLOT REFERENTIAL-STRUCTURES Output**

In the SQL 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 SQL Referential Structures Plot can be quite significant. The appearance of such a pointer serves to highlight one of two important situations in a SQL 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 the following diagram:

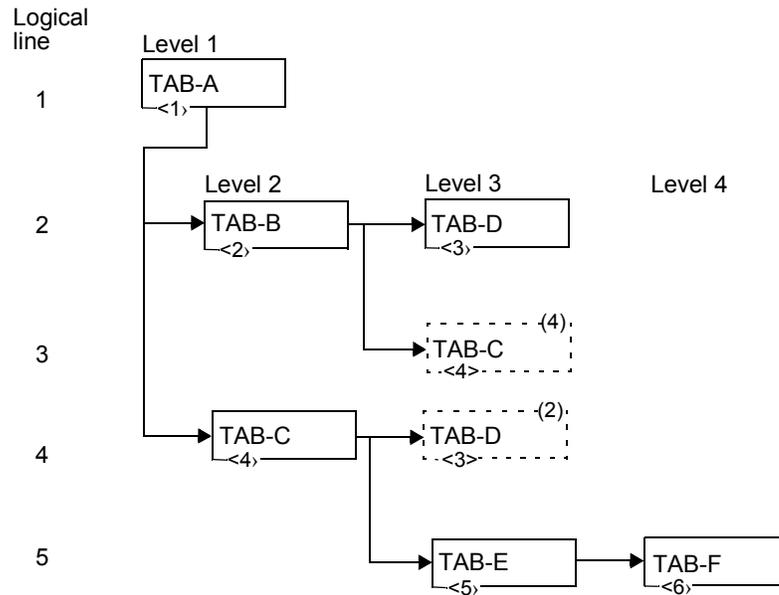


Figure 18 Use of Pointers in SQL Referential Structure Plots

In the sample plot, 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 appears 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 TAB-D is a dependent table of both TAB-B and TAB-C.
- No pointers are required for tables TAB-E and TAB-F because they are descendants of table TAB-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 a new 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 in the SQL PLOT REFERENTIAL-STRUCTURES Output**

After the main plot and any continuation plots have been displayed in the SQL Referential Structures Plot (and if the ALL keyword has been specified in the SQL PLOT command), ASG-DesignManager looks for any *additional seeds* that may be required to ensure that every table in the SQL 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.)

Refer to "Member Type and Command Descriptions" on page 143 for a discussion of how seeds are selected when ALL is specified in the command.

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.

Also, 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 SQL design.

*Use of Directories in the SQL PLOT REFERENTIAL-STRUCTURES Output*

At the end of the SQL 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 the 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 or table name.

The following diagram contains the same sample SQL Referential Structure Plot shown earlier in the discussion of the use of pointers. In addition, the corresponding Numeric and Alphabetic Directories are also given.

Refer to "Use of Pointers in the SQL PLOT REFERENTIAL-STRUCTURES Output" on page 67 for a discussion of the earlier diagram.

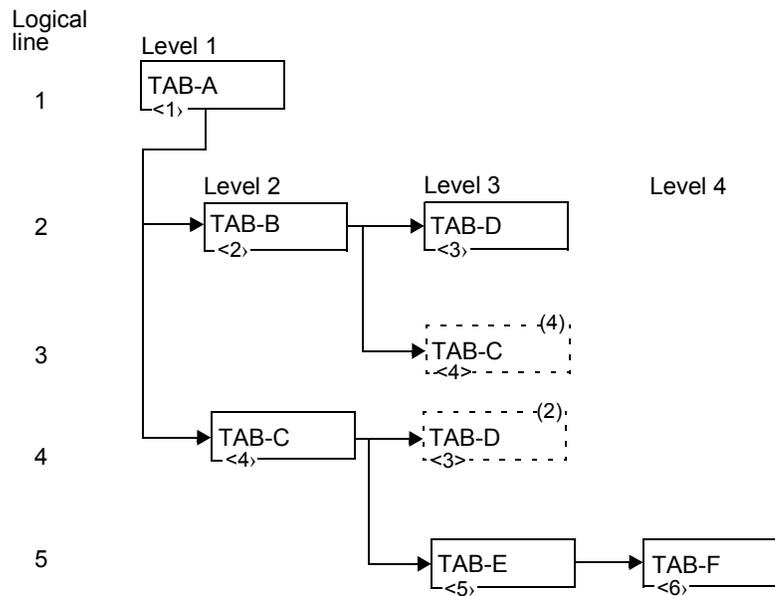


Figure 19 Use of Directories in an SQL Referential Structure Plot

**NUMERIC DIRECTORY**

|   | <b>Table</b> | <b>Line</b> | <b>Other Occurrences</b> |
|---|--------------|-------------|--------------------------|
| 1 | TAB-A        | 1           |                          |
| 2 | TAB-B        | 2           |                          |
| 3 | TAB-D        | 2           | 4                        |
| 4 | TAB-C        | 4           | 3                        |
| 5 | TAB-E        | 5           |                          |
| 6 | TAB-F        | 5           |                          |

**ALPHABETIC DIRECTORY**

|   | <b>Table</b> | <b>Line</b> | <b>Other Occurrences</b> |
|---|--------------|-------------|--------------------------|
| 1 | TAB-A        | 1           |                          |
| 2 | TAB-B        | 2           |                          |
| 4 | TAB-C        | 4           | 3                        |
| 3 | TAB-D        | 2           | 4                        |
| 5 | TAB-E        | 5           |                          |
| 6 | TAB-F        | 5           |                          |

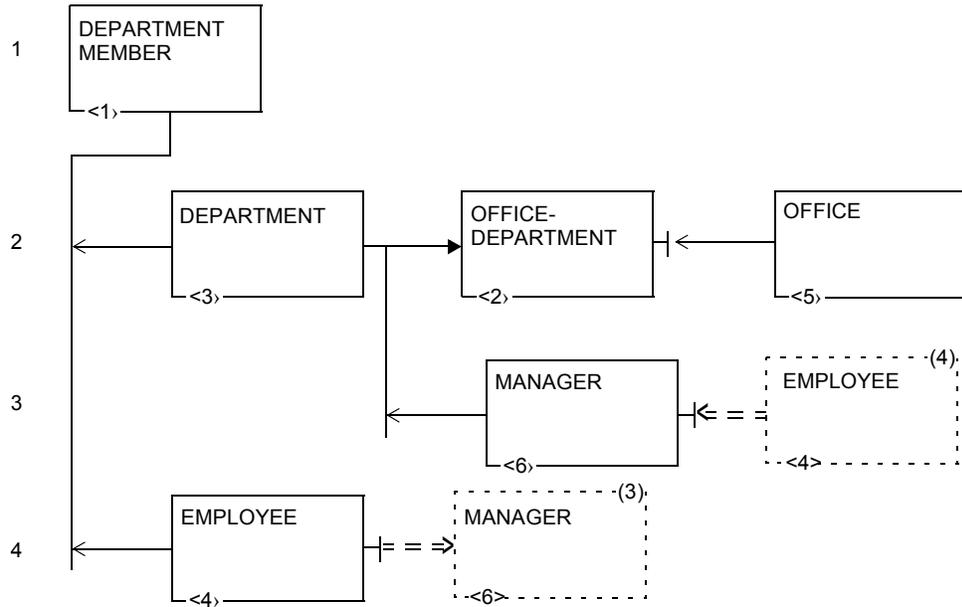
At a glance, you can see in the directories that tables TAB-C and TAB-D have entries in the third column, indicating that they both appear in more than one foreign key relationship.

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 of the SQL PLOT REFERENTIAL-STRUCTURE Output**

Following is an example of the output from the SQL 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.

```
*****
*          SQL REFERENTIAL-STRUCTURES PLOT          *
*****
```



**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                 |

**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    |                   |

```
*****
*           END OF SQL REFERENTIAL-STRUCTURES           *
*****
```

## Output from the SQL LIST TABLES Command

### Introduction to the SQL LIST TABLES Output

The SQL LIST TABLES command produces a list of all or some of the SQL tables in the Workbench Design Area (WBDA). For each SQL 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 of the SQL LIST TABLES Output

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

- The number of the table in the Workbench Design Area
- 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 SQL/DS, 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 SQL PLOT REFERENTIAL-STRUCTURES command.

### **Example of the SQL LIST TABLES Output**

The following is an example of output from the SQL LIST TABLES command for the Department Model:

**Table 9 List of SQL Tables Held in Workbench Design Area**

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

LIST CONTAINS 6 SQL TABLES

### **Output from the SQL LIST CYCLES Command**

#### **Introduction to the SQL LIST CYCLES Output**

The SQL LIST CYCLES command produces, for every cycle present in the SQL design generated in the Workbench Design Area (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.

Refer to "SQL LIST CYCLES" on page 197 for the definition of a cycle and of cyclic order.

### Description of the SQL LIST CYCLES Output

For each cycle found in the SQL design present in the Workbench Design Area (WBDA), the SQL 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 of the SQL LIST CYCLES Output

In the following diagram, an example is pictured of a cycle with its path of tables and connecting relationships:

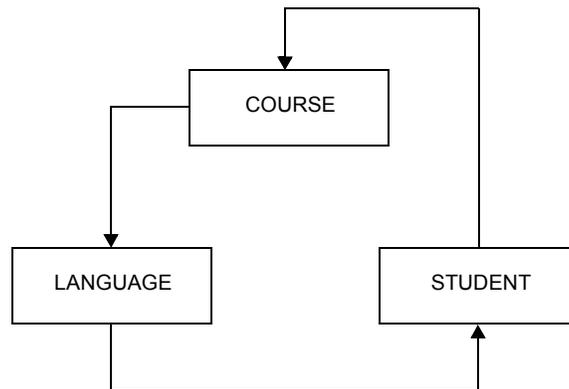


Figure 20 An Example of a Cycle in the SQL Design

The following output would be produced by the SQL 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 10 List of SQL Cycles Held in Workbench Design Area**

| <u>Cycle</u>                |             |          |
|-----------------------------|-------------|----------|
| Number                      | Key         | Names    |
| 1                           | COURSE-NO   | COURSE   |
| 2                           | LANGUAGE-NO | LANGUAGE |
| 3                           | STUDENT-NO  | STUDENT  |
| CYCLE CONTAINS 3 SQL TABLES |             |          |









```
➤——— ADD system-name —— ; ——➤
➤——— SYSTEM ——➤
➤——— <<, <<<<<
      CONTAINS sql-member ——➤
➤——— ; ——➤
```

where *sql-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.

Refer to "Generating and Previewing a SYSTEM Definition" on page 224 for details of generating SYSTEM members via the SQL PREVIEW command.

### **Example of Generated SYSTEM Dictionary Member**

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

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

---

# 4

## Dictionary Definition

---

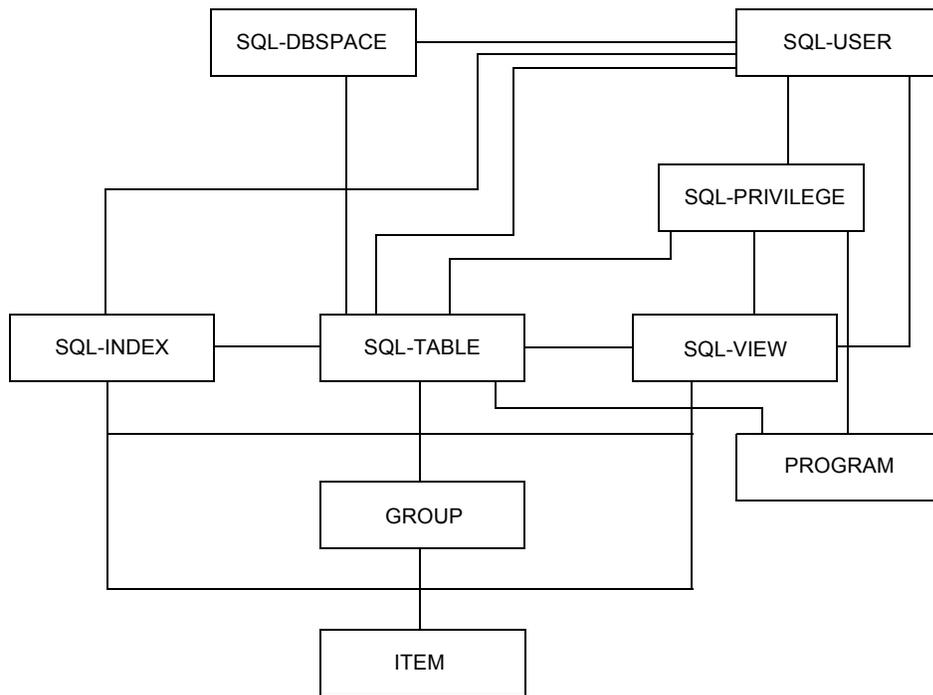


Figure 21 The Relationships between SQL/DS Member Types

In this diagram, the boxes represent the dictionary members which are used to document SQL/DS objects in the dictionary. The lines between them represent the relationships between the SQL/DS objects, which are established by clauses in the dictionary definition of the members which document them.

### Introduction to Documenting an SQL/DS DBMS

The Manager Products SQL/DS Definition Facility provides the dictionary member types which enable you to document SQL/DS objects. Generally, there is a one-to-one correspondence between an SQL/DS object available in the SQL/DS environment and the member in the dictionary which documents it.

You can share dictionary definitions of SQL/DS objects with other environments: the ITEM and GROUP members which document the columns of SQL/DS tables and views may also be used in other applications in an installation, and perhaps in other database schemas.

You can use the status facilities to reflect different stages in the life cycle of an SQL/DS database. The SQL/DS catalog cannot do this, since it records only the current status of the SQL/DS environment. Therefore, the dictionary can be an important tool for change control in an SQL/DS environment.

If you have the SQL/DS Database Design facility, you can populate your dictionary with SQL-TABLE, SQL-VIEW, and SQL-INDEX members from the first-cut SQL/DS design which is generated from the relational schema in the Workbench Design Area (WBDA). These members would constitute a first-cut dictionary schema, which you can go on to develop and complete in the dictionary.

Refer to *ASG-Manager Products Advanced Status* for information on the status facilities.

## Documenting SQL/DS Objects

SQL/DS objects in the SQL/DS database schema are documented in the dictionary as members of an equivalent type. The table below shows the correspondence between SQL/DS objects and the dictionary member types which are used to document them. An SQL/DS dictionary member can be said to represent the corresponding SQL/DS object.

**Table 11 Correspondence between Members and SQL/DS Objects**

| <b>Dictionary</b> | <b>SQL/DS Object</b> |
|-------------------|----------------------|
| SQL-DBSPACE       | dbspace              |
| SQL-INDEX         | index                |
| SQL-TABLE         | table                |
| SQL-VIEW          | view                 |
| ITEM (or GROUP)   | column               |

A further 2 member types are available which enable you to document the Authorization ID of an SQL/DS user, and the privileges held by users. They are the SQL-USER and SQL-PRIVILEGE member types. You can use these 2 member types to model the SQL/DS security system. They are also the member types from which SQL/DS access privileges are generated as SQL GRANT and REVOKE statements.

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

In order that you do not have to duplicate 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 which are common to more than one member, are entered only once in the dictionary, and only one member needs to be maintained.

It is important to note that the only clause which 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.

Certain checks are made when a member is encoded, usually to ensure that members referred to are of the correct type and that external names conform to the requirements of the target external environment. The checks which are made on each clause are indicated in the section documenting each member type.

Certain clauses must be present in an SQL/DS member definition when you generate an SQL statement from it for generation to be successful. The clauses which must be present for the successful generation of any particular SQL statement are indicated in the section documenting each member type.

Refer to "Documenting SQL/DS Security Information" on page 88 for further information on support for the SQL/DS security system.

Refer to "Defining an AS Clause" on page 277 for details of the use of the AS clause.

## Clauses Establishing Relationships between SQL/DS Member Types

The following table shows the relationships which are possible between SQL/DS member types, and the clauses which document these relationships.

**Table 12 Clauses which Create Relationships between Members**

|             | SQL-DBSPACE | SQL-TABLE  | SQL-VIEW | SQL-USER      | ITEM/GROUP | PROGRAM |
|-------------|-------------|------------|----------|---------------|------------|---------|
| SQL-DBSPACE |             |            |          | CREATOR-OWNER |            |         |
| SQL-TABLE   | IN          | REFERENCES |          | CREATOR-OWNER | CONTAINS   |         |
| SQL-VIEW    |             | FROM       | FROM     | CREATOR-OWNER | CONTAINS   |         |
| SQL-INDEX   |             | ON         |          | CREATOR-OWNER | CONTAINS   |         |

**Table 12 Clauses which Create Relationships between Members**

|               |  |          |          |               |          |    |
|---------------|--|----------|----------|---------------|----------|----|
| SQL-USER      |  | SYNONYMS | SYNONYMS |               |          |    |
| SQL-PRIVILEGE |  | ON       | ON       | GRANTOR<br>TO | CONTAINS | ON |
| PROGRAM       |  | SEE      |          |               |          |    |

The member types listed along the top of the table are referred to by the member types listed down the left-hand column via the clauses entered against them in the table. For example, the IN clause in the SQL-TABLE member type establishes a relationship with the SQL-DBSPACE member type.

## Documenting the Columns of SQL/DS Tables and Views

Use ITEM and GROUP member-types to represent the columns of SQL/DS tables and views. In ITEMS and GROUPs you can document the types of data which can be held in the columns which they represent. These ITEMS and GROUPs are referred to by SQL-TABLE, SQL-VIEW, SQL-INDEX, and SQL-PRIVILEGE members when SQL statements and host language data structures are generated in SQL CREATE, SQL ALTER, SQL PRODUCE, and SQL GRANT commands.

The data type of a generated column is determined firstly by the form keyword and version specified in the clause in which columns are specified in an SQL-TABLE or SQL-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 column definition clause applies to *all* the ITEMS and GROUPs specified in the clause. If the first version of the specified form of any individual ITEM does not hold the correct data type, you can specify for that ITEM a different version (of the same form) which does hold the correct data type.

When PL/I, COBOL, or Assembler host language data structures are generated from an ITEM which includes any of the following USAGE clauses:

- USAGE TIME
- USAGE DATE
- 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 SQL/DS.

In ITEMS which 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 CM\_LOCAL\_TIME and CM\_LOCAL\_DATE.

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

The SQL data type of a column generated from a GROUP specified in the single-column clause depends on the aggregate length and whether any of the contained ITEMS are of variable length. It is one of the following:

|                      |   |
|----------------------|---|
| CHAR( <i>m</i> )     | Where <i>m</i> is the aggregate length of fixed length fields which constitute the group and <i>m</i> is less than 255 characters.  |
| VARCHAR ( <i>m</i> ) | Where <i>m</i> is the aggregate length of fields which constitute the group and one or more of the fields are of variable length and <i>m</i> is less than 255 characters.            |
| LONG VARCHAR         | Where <i>m</i> is the aggregate length of fields which constitute the group (regardless of whether none or some are of variable length) and <i>m</i> is greater than or equal to 255. |

To find out the length of the aggregate fields, enter:

```
PRODUCE RECORD-LAYOUT FROM group-name;
```

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

**Note:** \_\_\_\_\_

The data types of the columns generated will be expressed differently according to the destination language (SQL, PL/I, COBOL, or Assembler).

**Table 13 Documenting the Data Type of Columns in ITEM Members**

| ITEM USAGE Clause | ITEM Form-Description     | Data Types by Language:          |                 |                        |           |
|-------------------|---------------------------|----------------------------------|-----------------|------------------------|-----------|
|                   |                           | SQL                              | PL1             | COBOL                  | PL1       |
| TIME              | CHAR t                    | TIME                             | CHAR (t)        | PIC X(t)               | DS CLt    |
|                   | Other                     | Accepted but should not be used. |                 |                        |           |
| TIME STAMP        | CHAR 26                   | 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<br>p = 1..127      | GRAPHIC(p)                       | GRAPHIC (p)     | PIC G(p)<br>DISPLAY-1. | DS CL2p   |
|                   | CHAR p TO q<br>p = 1..127 | VARGRAPHIC (q)                   | GRAPHIC (q) VAR | 10 x.<br>49 x-L PIC    | DS H.CL2q |

**Table 13 Documenting the Data Type of Columns in ITEM Members**

| ITEM USAGE Clause | ITEM Form-Description  | Data Types by Language: |                       |                             |                |
|-------------------|--|-------------------------|-----------------------|-----------------------------|----------------|
|                   |  | SQL                     | PL1                   | COBOL                       | PL1            |
|                   | CHAR p<br>p > 127  | LONG<br>VARGRAPHIC      | GRAPHIC<br>(p) VAR    | S9 (4) COMP.<br>49 x-D PIC  | DS<br>H.CL2p   |
|                   | CHAR p TO q<br>p > 127   |                         | GRAPHIC<br>(q) VAR    | G (q/p)<br>DISPLAY-1.       | DS<br>H.CL2q   |
|                   | Other  | Error.                  |                       |                             |                |
| MONEY             | Form-description is used as when no USAGE clause present.      |                         |                       |                             |                |
| POINTER           | Any  | Error.                  |                       |                             |                |
| None              | BIN p<br>BIN p TO q<br>BIN n.m<br>p, q, or n+m<br>= 1..4       | SMALLINT                | FIXED BIN<br>(15)     | PIC S9 (4) COMP             | DS H           |
| None              | BIN p<br>BIN p TO q<br>BIN n.m<br>p, q, or n+m<br>= 5..9       | INTEGER                 | FIXED BIN<br>(31)     | PIC S9 (9) COMP             | DS F           |
| None              | BIN p<br>BIN p TO q<br>BIN n.m<br>p, q, or n+m<br>> 9          | FLOAT (21)              | FLOAT BIN<br>(21)     | COMP-1                      | DS E           |
| None              | DEC n.m<br>n+m = 1..15   | DECIMAL<br>(n+m, m)     | FIXED DEC<br>(n+m, m) | PIC S9 (n) V9 (m)<br>COMP-3 | DS<br>PLc'a.b' |
| None              | DEC p<br>p = 1..15   | DECIMAL (p)             | FIXED DEC<br>(p)      | PIC S9 (p)<br>COMP-3        | DS<br>PLc'a'   |
| None              | DEC p TO q<br>q = 1..15  | DECIMAL (q)             | FIXED DEC<br>(q)      | PIC S9 (q)<br>COMP-3        | DS<br>PLc'b'   |
| None              | DEC p<br>DEC p TO q<br>DEC n.m<br>p, q, or n+m<br>> 15         | Error.                  |                       |                             |                |
| None              | FLOAT p<br>FLOAT p TO q<br>FLOAT n.m<br>p, q, or n+m<br>= 1..6 | FLOAT (21)              | FLOAT BIN<br>(21)     | COMP-1                      | DS E           |

**Table 13 Documenting the Data Type of Columns in ITEM Members**

| ITEM USAGE Clause | ITEM Form-Description  | Data Types by Language: |                          |                            |                  |
|-------------------|--|-------------------------|--------------------------|----------------------------|------------------|
|                   |  | SQL                     | PL1                      | COBOL                      | PL1              |
| None              | FLOAT <i>p</i><br>FLOAT <i>p</i> TO <i>q</i><br>FLOAT <i>n.m</i><br><i>p</i> , <i>q</i> , or <i>n+m</i><br>> 6 | FLOAT (53)              | FLOATBIN (53)            | COMP-2                     | DS D             |
| None              | CHAR <i>p</i><br><i>p</i> = 1..254   | CHAR ( <i>p</i> )       | CHAR ( <i>p</i> )        | PIC X( <i>p</i> )          | DS CL <i>p</i>   |
| None              | CHAR <i>p</i> TO <i>q</i><br><i>p</i> = 1..254   | VARCHAR ( <i>q</i> )    | CHAR ( <i>q</i> )<br>VAR | 10 x.<br>49 x-L PIC        | DS H,CL <i>q</i> |
| None              | CHAR <i>p</i><br><i>p</i> > 254  | LONG<br>VARCHAR         | CHAR ( <i>p</i> )<br>VAR | S9 (4) COMP.<br>49 x-D PIC | DS H,CL <i>p</i> |
| None              | CHAR <i>p</i> TO <i>q</i><br><i>p</i> > 254  |                         | CHAR ( <i>q</i> )<br>VAR | X( <i>q/p</i> ).           | DS H,CL <i>q</i> |
| None              | Other  | Error.                  |                          |                            |                  |

A PICTURE clause in an ITEM member definition is interpreted as a form-description of CHARACTER *p*, where *p* is the number of bytes which are generated from the PICTURE clause when a PRODUCE RECORD-LAYOUTS command is applied to the member. A column with a data type of CHAR *p*, LONG VARCHAR, GRAPHIC *p*, LONG VARGRAPHIC, TIME, TIMESTAMP, or DATE will (depending on the value of *p* and whether the ITEM has a USAGE clause) be generated from the ITEM.

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.

*p* and *q* are integers.

$t$  is a character field length with a default of 8 and  $d$  is a character field length with a default of 10. 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 reducing the data name (using the Name Reduction Process) to 30 characters so as not to exceed the COBOL limit for name lengths.

' $c$ ' 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 9s. The number of 9s is equal to the value of  $p$  or  $n$ .

$b$  is a sequence of 9s. The number of 9s is equal to the value of  $q$  or  $m$ .

Refer to "Introduction to the Name Reduction Process" on page 279 for details of the name reduction process.

## Documenting SQL/DS Security Information

The SQL/DS security system enables you to control access to objects in the SQL/DS environment. In order that you can document the SQL/DS security system we have provided the following member types in the dictionary schema:

- SQL-USER
- SQL-PRIVILEGE.

An SQL/DS user must have an Authorization ID to be able to sign on to SQL/DS. This Authorization ID is recorded in the dictionary as an SQL-USER member type.

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

All privileges are granted by a particular user (the GRANTOR) to another user (the GRANTEE).

Together, the SQL-USER and SQL-PRIVILEGE member types allow you to document the SQL/DS objects to which particular users have access, and their access rights. These members can be interrogated by dictionary interrogation commands to provide Database Administrators with the ability to analyze the dictionary model of the SQL/DS security system.

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

## **Naming Conventions for SQL/DS Members**

### ***The Derivation of External Names from SQL/DS Members***

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

In addition, when you generate COBOL, PL1, or Assembler host language data structures (by the SQL PRODUCE command) from an SQL/DS member which contains 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, PL1, or Assembler.

In order to generate an acceptable external name, a number of processes are applied. Therefore, when you document an SQL/DS member in the dictionary, it is important to know how the external name will be derived from the dictionary definition.

If you want to be able to generate external names from aliases, your Systems Administrator must ensure that your SQL/DS profile is tailored appropriately, and that an Alias Table exists for each language you want to generate.

In all external names, except synonyms, 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 the Name Reduction Process takes place.

Refer to "Introduction to the Name Reduction Process" on page 279 for details of the Name Reduction Process.

### ***The Derivation of Column Names from SQL/DS Members***

The names of columns in tables and indexes are derived from one of the sources listed below. If an acceptable name is not found in the first source, it will be sought in the second, and so on.

- The first source is the name specified in the KNOWN-AS clause of the SQL-TABLE or SQL-INDEX in which the columns are defined
- The second source is an alias of the ITEM or GROUP member which represents the column (assuming your environment is tailored to generate aliases as external names). Names derived from this source are checked by the Name Reduction Process.
- The third source is the dictionary name of the ITEM or GROUP member which represents the column. Names derived from this source are checked by the Name Reduction Process.

The above conditions apply to the columns in a view, unless a column-name is specified in the COLUMN-NAME clause of the SQL-VIEW member which represents the view. The name specified there will override any KNOWN-AS name, alias, or member name.

The unqualified SQL/DS name of a column may be no longer than 18 characters.

### ***The Derivation of the Names of Tables, Views, Indexes, and Dbspaces from SQL/DS Members***

The names of tables, views, indexes, and dbspaces are derived from one of the sources listed below. If an acceptable name is not found in the first source, it will be sought in the second.

- The first source is an alias name specified in the members which represent them (assuming your environment is tailored to generate aliases as external names)
- The second source is the dictionary names of the members which represent them. Names derived from this source are checked by the Name Reduction Process.

The unqualified SQL/DS object names for tables, views, indexes, and dbspaces may be no longer than 18 characters.

### ***The Derivation of the Names of Synonyms, Constraints, Correlations, and Programs***

The SQL/DS object names of synonyms, constraints, correlations, and programs are derived directly from the corresponding names specified in the dictionary definitions where they appear. The names of synonyms, constraints, and correlations may be no longer than 18 characters. The names of programs may be no longer than 8 characters.

**Note:** \_\_\_\_\_

On generation of an SQL statement, a hyphen in a synonym-name specified in a dictionary member is not translated into an underscore. A hyphen in a constraint-name, correlation-name, or program-name is translated into an underscore.

\_\_\_\_\_

## The Derivation of SQL/DS User Names

The user name used to qualify the SQL/DS names of tables, views, indexes, and dbspaces is derived from one of the sources listed below. If an acceptable name is not found in the first source, it will be sought in the second.

- The first source is an alias of the SQL-USER member referred to in the CREATOR-OWNER clause of the member representing the object being qualified (assuming your environment is tailored to generate aliases as external names)
- The second source is the dictionary name of the SQL-USER referred to in the CREATOR-OWNER clause of the member representing the object being qualified.

An SQL/DS user name (that is an Authorization ID) may be no longer than 8 characters.

## Naming Guidelines for SQL/DS Members

Although mechanisms are provided to derive names for SQL/DS objects which will be acceptable when SQL/DS 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 may not necessarily know the name of the dictionary member which documents an SQL/DS object. This is the case when names are reduced by the Name Reduction Process and when you specify names in the KNOWN-AS clauses of SQL-TABLE, SQL-VIEW, SQL-PRIVILEGE, and SQL-INDEX member definitions. When such a loss of information does take place, it may be no longer possible to determine unambiguously the dictionary origin of an external name.

It is important, therefore, to establish a naming strategy before you start to define SQL/DS members in your dictionary.

The simplest approach to a reliable naming strategy is to ensure that all external names of SQL/DS objects are derived directly from the dictionary member name of the member which represents it in the dictionary.

You would not be able to define column-names (defined in COLUMN-NAME clauses) in SQL-VIEWS, or known-as names (defined in KNOWN-AS clauses) in SQL-TABLES, SQL-VIEWS, SQL-PRIVILEGES, and SQL-INDEXES. Nor would you be able to define alias names (in ALIAS clauses) for any member representing a DB2 object.

A more efficient alternative to omitting ALIAS clauses, is to de-activate the ALIAS searching mechanism, since the extra work involved in deriving external names from aliases can increase processing time, by up to three times.

Since the external name will be taken from the dictionary name of the member which represents an SQL/DS object, there will be a direct correspondence between the dictionary and external environments. Therefore, it will be straightforward to match SQL/DS objects with the dictionary members which represent them, since they will have the same name in the dictionary as they do in the SQL/DS catalog, and in host language data 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 SQL/DS 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 such a situation, you should consider using aliases which 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 dictionary (which means that two unrelated members may have the same alias), interrogation commands which can detect a duplicate alias can be used to guard against alias duplication.

For instance, the dictionary interrogation:

```
WHOSE SQL ALIAS IS 'EMP-MAIN-TAB' ;
```

returns all members of the type SQL-TABLE which have this alias. You can then change the alias names of all but one of these SQL-TABLE members, in order to achieve unique names. The table name 'EMP\_MAIN\_TAB' in the SQL/DS catalog would then correspond with the SQL ALIAS of one and only one SQL-TABLE dictionary member.

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

In conclusion, think carefully about your choice of dictionary and external names. If necessary, use aliases and interrogation capabilities to the full to implement a sound naming strategy which will allow for a direct correspondence between your SQL/DS dictionary schema, the database schema which it represents, and other external environments in which objects may be used.

## **Processing Your SQL/DS Members**

You can process and interrogate your SQL/DS dictionary schema in the usual way using Manager Products dictionary management commands.

The interrogation keywords:

- SQL-DBSPACE
- SQL-INDEX
- SQL-PRIVILEGE
- SQL-TABLE
- SQL-USER
- SQL-VIEW

are added to the member-type keywords available for use in the following commands:

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

In addition, the alias-type keyword SQL is available in the ALIAS clause. If you define an SQL ALIAS for an SQL/DS member, you can ensure that the name you define will be used in the SQL/DS environment.

You can use all the clauses which establish relationships between SQL/DS members as keywords in the VIA clause of the WHICH and WHAT commands. These clauses are all of those which establish relationships between members, and the AS clause.

For example, to find out the ITEMS and GROUPs which constitute the SQL-TABLE, EMP-TABLE, enter the following command:

```
WHICH ITEMS, GROUPS DIRECTLY CONSTITUTE EMP-TABLE VIA CONTAINS ;
```

You can use all the clauses of SQL/DS members as interrogation keywords with the FOR clause of the GLOSSARY command and the HAS/HAVE clause of the WHICH command.

For example, to find out the creator-owner of every SQL-TABLE in a dictionary, enter the following command:

```
GLOSSARY FOR SQL-TABLE GIVING CREATOR-OWNER ;
```

To find out which SQL-PRIVILEGE members document SYSTEM privileges, enter the following command:

```
WHICH SQL-PRIVILEGES HAVE SYSTEM SPECIFIED
```

Refer to the *ASG-Manager Products Dictionary/Repository Guide* for details of the GLOSSARY, WHAT, and WHICH commands.



# 5

## Implementation and Maintenance

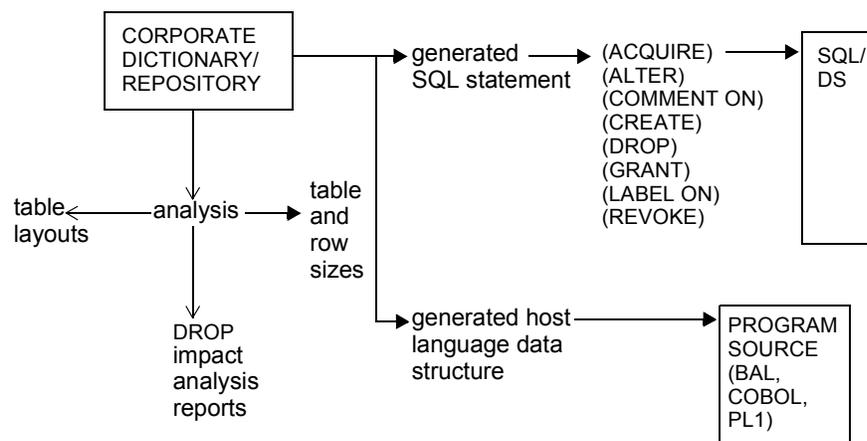


Figure 22 Exporting SQL Statements and Data Structures

### Introduction to Generating SQL Statements and SQL/DS Host Language Data Structures

#### Overview of Generating SQL Statements and SQL/DS Host Language Data Structures

The Corporate Dictionary/Repository Definition Export for SQL/DS facility enables you to generate SQL statements and host language data structures from the definition of members in the dictionary. You can also calculate the size of tables and their rows.

The dictionary members concerned are those provided by the SQL/DS Definition facility to document in the dictionary the objects which exist or are to be created in your SQL/DS environment.

You can automatically file generated output in a USER-MEMBER and subsequently transfer it to an external file which can be used as input to your SQL/DS environment. You can also use Dynamic SQL Services to dynamically submit the SQL statements you have generated, to your SQL/DS environment, from within Manager Products.

Your SQL/DS environment can therefore be documented in the dictionary and implemented and maintained with output generated from the dictionary.

The SQL ACQUIRE, SQL ALTER, SQL COMMENT, SQL CREATE, SQL DROP, SQL GRANT, SQL LABEL, SQL REVOKE, and SQL SYNONYM commands generate the following SQL statements:

|            |                |
|------------|----------------|
| ACQUIRE    | GRANT          |
| ALTER      | LABEL ON       |
| COMMENT ON | REVOKE         |
| CREATE     | CREATE SYNONYM |
| DROP       | DROP SYNONYM   |

The SQL PRODUCE command generates an Assembler, COBOL, or PL1 host language data structure. These are for use, in conjunction with embedded SQL statements, within application programs. You can also use the SQL PRODUCE command to generate table layouts which document the column structure of tables and views in a tabular format.

You can analyze the dictionary before applying the SQL statements you have generated to your SQL/DS environment:

- The SQL DROP command generates both an SQL DROP statement and a report showing the impact of that statement
- The SQL SIZE command calculates the size of a table and its rows. You can therefore estimate the number of pages required in a dbspace before generating an SQL ACQUIRE DBSPACE statement.

The Systems Administrator can tailor the SQL statements and host language data structures generated to suit your environment. For example SQL/DS object names in SQL statements and external names in host language data structures can be derived from aliases instead of member names. If your installation uses the SQL and COBOL languages you could specify that SQL/DS object names will be derived from SQL aliases and external names from COBOL aliases.

Refer to Chapter 7, "Dynamic SQL Services," on page 107 for details of Dynamic SQL Services.

## **Generating Column Data Types**

You can generate data types for columns in a table and for column variables in a host language data structure.

The SQL ALTER and SQL CREATE commands generate SQL statements specifying the SQL/DS data type of columns in tables. The SQL PRODUCE command generates Assembler, COBOL, or PL1 host language data structures for tables and views with column variables corresponding to the SQL/DS data types.

The data types are derived from the definition of the ITEMS and/or GROUPs specified in the CONTAINS clause of the SQL-TABLE or SQL-VIEW member from which the SQL statement or host language data structure is generated.

Refer to "Documenting the Columns of SQL/DS Tables and Views" on page 84 for details of how the data type of columns and column variables are derived from ITEMS and GROUPs.

## Submitting Generated Output to Your Relational Environment

You will want to submit to your relational environment the SQL statements and host language data structures you have generated using the D32 or SQL commands.

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

Alternatively you can file the generated output in a USER-MEMBER by specifying an ONTO clause in a DB2 or SQL command, and then use the TRANSFER command to transfer the output from the USER-MEMBER into an external file. The external file can be used as input to your relational environment or for inclusion in your application programs.

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

SQL DECLARE statements and host language data structures are intended to be imbedded in application programs and must be transferred to an external file by a TRANSFER command. Your application program can then use SQL COPY or INCLUDE statements to reference the external file.

Refer to Chapter 7, "Dynamic SQL Services," on page 107 for details of Dynamic SQL Services.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

## Tailoring SQL Statements and SQL/DS Host Language Data Structures

### Introduction to Tailoring

#### How to Tailor Generated Output

The Systems Administrator can create a *SQL/DS profile* which will tailor the SQL statements and host language data structures generated.

The SQL/DS profile is an *Executive Routine* containing *SET directives* which by assigning alternative values to variables allows you to control how output is generated. For example, SQL/DS object names in SQL statements can be derived from aliases instead of member names.

You must first create and execute an Executive Routine which with the following directives declares a *Global Variable* set to the name of the SQL/DS profile:

```
GLOBAL GL_PROFILENAME  
SET GL_PROFILENAME :sql/ds-profile:
```

where *sql/ds-profile* is the name of the SQL/DS profile Executive Routine.

SQL/DS profiles can be set up for individual or all users whenever they logon, by creating the Executive Routine which declares the Global Variable as a *User Defined Profile*, or by executing the Executive Routine from within a *Logon Profile* or *Global Profile*.

If you wish to derive SQL/DS object names in SQL statements and external names in host language data structures from aliases you must also create an *Alias Table* for each of the languages that can be generated (Assembler, PLI, COBOL, and SQL). Alias Tables are Executive Routines containing SET directives.

An EXECUTIVE member named MPR-EX-PRF is provided as an example of a SQL/DS profile. Four EXECUTIVE members named MPR-EX-BAL, MPR-EX-PLI, MPR-EX-COB, and *MPR-EX-DB2* are provided as examples of Alias Tables for the Assembler, PLI, COBOL, and SQL languages. These EXECUTIVE members can be used as models.

All users who create their own SQL/DS profile in a User Executive Routine should do so under the supervision of the Systems Administrator.

Refer to "Glossary" on page 283 for a brief description of the terms used in this introduction.

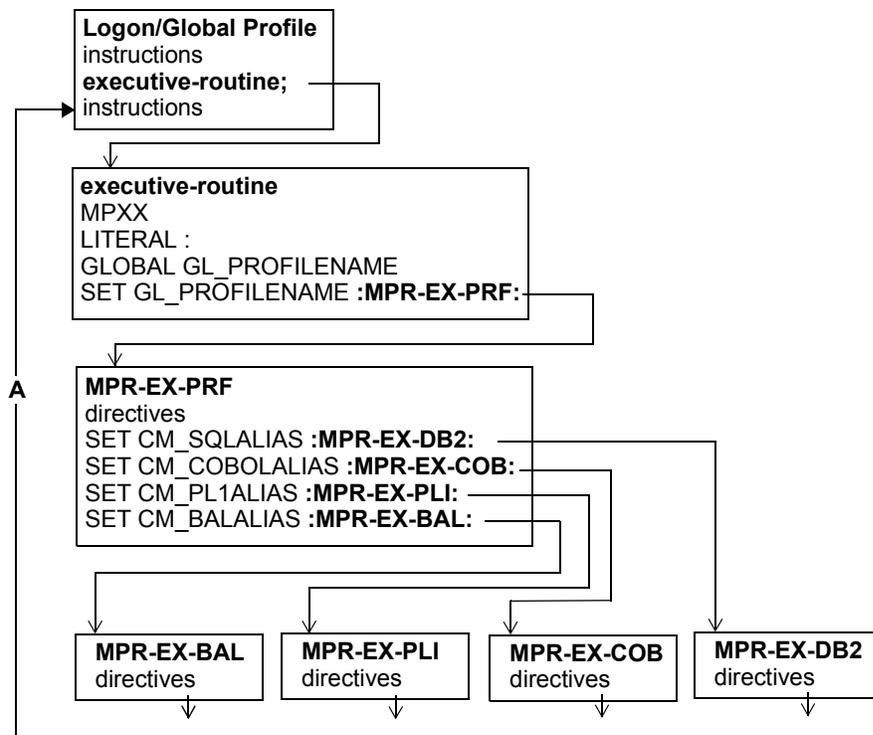


Figure 23 How to Tailor SQL Statements and Data Structures

The illustration includes the EXECUTIVE members MPR-EX-PRF, MPR-EX-DB2, MPR-EX-COB, MPR-EX-PLI, and MPR-EX-BAL provided by ASG as examples.

**The Directives that Tailor SQL Statements and Host Language Data Structures**

By entering particular directives in your SQL/DS profile you can tailor the SQL statements and host language data structures generated.

To declare a Global Variable identifying the SQL/DS profile:

```
GLOBAL GL_PROFILENAME and SET GL_PROFILENAME
```

To generate SQL CREATE and COMMENT ON statements at the same time:

```
SET CM_COMMENTOPT
```

To generate SQL CREATE and LABEL ON statements at the same time:

```
SET CM_LABELOPT
```

To generate a host language indicator structure:

```
SET CM_INDICATOROPT
```

To generate column variables in host language data structures that are compatible with your SQL/DS installation settings for date:

```
SET CM_LOCAL_DATE
```

To generate column variables in host language data structures that are compatible with your SQL/DS installation settings for time:

```
SET CM_LOCAL_TIME
```

To display internal diagnostic output:

```
SET CM_DEBUG
```

To derive SQL/DS object names in SQL statements from aliases:

```
SET CM_SQLALIAS
```

To derive external names in structures from aliases:

```
SET CM_BALIAS
```

To derive external names in from aliases:

```
SET CM_COBOLALIAS
```

To derive external names in from aliases:

```
SET CM_PLIALIAS :
```

To specify the type of alias from which SQL/DS object names and external names will be derived:

```
SET CM_ALIAS (n)
```

To stop SQL/DS object names and external names being derived from aliases by deactivating Alias Tables:

```
SET CM_ALIASNUM 0
```

### **Displaying Internal Diagnostic Output**

You can tailor the output of the SQL ACQUIRE, SQL ALTER, SQL COMMENT, SQL CREATE, SQL DROP, SQL GRANT, SQL LABEL, SQL PRODUCE, SQL REVOKE, SQL SIZE, and SQL SYNONYM commands so that internal diagnostic output is printed.

It may sometimes be necessary for ASG to request a print-out of this internal diagnostic output in order to satisfy a request for maintenance. There are two levels of internal diagnostic output, either or both of which can be printed.

To display level 1 diagnostic output, enter the following SET directive in your SQL/DS profile:

```
SET CM_DEBUG :DEBUG1 :
```

To display level 2 diagnostic output, alter the SET directive to read:

```
SET CM_DEBUG :DEBUG2 :
```

To display both level 1 and level 2 diagnostic output, alter the SET directive to read:

```
SET CM_DEBUG :DEBUG3 :
```

**Level 1 diagnostic output** is a list of the variables and values initially loaded from member definitions for use in generation. It precedes normal output generated by the command.

**Level 2 diagnostic output** is a list of these variables and values as they have been subsequently generated in an SQL statement or host language data structure. It follows normal output generated by the command.

For example if a local name, member name, or alias exceeds the length permitted for SQL/DS object names or external names it will, when generated, be reduced in length by the name reduction process. The full name will be shown in level 1 and the reduced name in level 2 diagnostic output.

Only level 1 diagnostic output is printed by the SQL SIZE command as it does not generate SQL statements or host language data structures.

Internal diagnostic output is not filed on the MP-AID when an ONTO clause is specified in a command. The NOPRINT keyword will not prevent internal diagnostic output being printed.

To return to the default in which no diagnostic output is printed, either remove the relevant SET directive or alter it to read:

```
SET CM_DEBUG :OFF :
```

Refer to "Introduction to the Name Reduction Process" on page 279 for details of the name reduction process.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause and NOPRINT keyword.

## **Generating Object Names and External Names from Aliases**

### **Deriving SQL/DS Object Names and External Names from Aliases**

You can tailor the output of the SQL ACQUIRE, SQL ALTER, SQL COMMENT, SQL CREATE, SQL DROP, SQL GRANT, SQL LABEL, SQL PRODUCE, SQL REVOKE, and SQL SYNONYM commands so that SQL/DS object names in SQL statements and external names in host language data structures are derived from aliases.

To generate aliases as SQL/DS object names in SQL statements, enter the following SET directive in your SQL/DS profile:

```
SET CM_SQLALIAS :sql-alias-table:
```

To generate aliases as external names in Assembler, COBOL, and PLI host language data structures, enter the following SET directives in your SQL/DS profile:

```
SET CM_BALIAS :assembler-alias-table:
```

```
SET CM_COBOLALIAS :cobol-alias-table:
```

```
SET CM_PLIALIAS :pli-alias-table:
```

where *sql-alias-table*, *assembler-alias-table*, *cobol-alias-table*, and *pli-alias-table* are Executive Routines containing SET directives which specify the types of alias from which the SQL/DS object or external name will be derived.

A separate Alias Table should be created for each language that can be generated (Assembler, COBOL, PLI, and SQL). In each Alias Table you specify the types of alias from which you want SQL/DS object or external names to be derived when generating that language.

For example, when generating COBOL host language data structures you may want to derive external names from COBOL aliases, but when generating SQL statements to derive SQL/DS object names from SQL aliases. To do so you would specify the COBOL alias type in the *cobol-alias-table* and the SQL alias type in the *sql-alias-table*.

### **Deriving SQL/DS Object Names and External Names from Particular Types of Aliases**

You can derive SQL/DS object names in SQL statements and external names in host language data structures from particular types of alias.

To specify the type of alias from which SQL/DS object or external names will be derived, enter the following SET directive in each Alias Table:

```
SET CM_ALIAS(n) :alias-type:
```

where *alias-type* is any of the types of aliases available in your dictionary. A separate SET directive must be entered for each alias-type.

Use the SHOW ALIAS-TYPES command to find out the types of alias available in your dictionary.

Alternative keywords (alias-type synonyms) can be specified in the DALIAS installation macro for each alias-type. SQL/DS object names and external names are only generated correctly when you use the first alias-type keyword specified in the DALIAS macro.

Refer to your installation manual for details of the DALIAS macro.

where *n* is an integer specifying the sequence in which each alias-type will be interrogated to derive the SQL/DS object or external name, if more than one SET directive has been entered.

The SQL/DS object or external name will be derived from the first of the interrogated alias-types found in the encoded record of the member from which the output is being generated.

Alias-type 1 will be interrogated first, and if it is not present, alias-type 2 is interrogated next and so on until an alias-type which is interrogated is found in the members encoded record.

If alias-type is NO-TYPE the SQL/DS object or external name will be derived from the first general alias interrogated. You can only derive the SQL/DS object or external name from the first general alias in the members encoded record.

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

Four EXECUTIVE members named MPR-EX-BAL, MPR-EX-PLI, MPR-EX-COB, and MPR-EX-DB2 are provided as examples of Alias Tables for the Assembler, PLI, COBOL, and SQL languages and can be used as models.

To return to the default either remove the relevant directives or enter the SET directive:

```
SET CM_ALIASNUM 0
```

in a particular Alias Table to deactivate that Alias Table, or in the SQL/DS profile to deactivate all the Alias Tables.

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

### **Generating SQL CREATE, LABEL ON, and COMMENT ON Statements from One Member at the Same Time**

You can tailor the output of the SQL CREATE command so that SQL CREATE, COMMENT ON, and/or LABEL ON statements are generated from the same member at the same time.

To generate SQL CREATE and COMMENT ON statements enter the following SET directive in your SQL/DS profile:

```
SET CM_COMMENTOPT :ON:
```

To generate SQL CREATE and LABEL ON statements, enter the following SET directive in your SQL/DS profile:

```
SET CM_LABELOPT :ON:
```

Processing times are faster if these SQL statements are generated at the same time rather than with separate SQL CREATE, SQL COMMENT, and SQL LABEL commands.

You can still generate only SQL COMMENT ON or LABEL ON statements with the SQL COMMENT and SQL LABEL commands.

To return to the default in which separate SQL CREATE, SQL COMMENT, and SQL LABEL commands are required to generate SQL CREATE, COMMENT ON, and LABEL ON statements, either remove the relevant SET directives or alter them to read:

```
SET CM_COMMENTOPT :OFF:
```

```
SET CM_LABELOPT :OFF:
```

Refer to "Member Type and Command Descriptions" on page 143 for an example of SQL CREATE, COMMENT ON, and LABEL ON statements generated from an SQL-TABLE member definition.

Refer to "Member Type and Command Descriptions" on page 143 for details of the SQL CREATE, SQL COMMENT, and SQL LABEL commands.

### **Generating a Host Language Indicator Structure**

You can tailor the output of the SQL PRODUCE command so that a host language indicator structure is generated.

To generate a host language indicator structure, enter the following SET directive in your SQL/DS profile:

```
SET CM_INDICATOROPT :ON:
```

Indicator structures consist of an array of half word binary variables with as many elements as there are columns in the table. The indicator variables can be referenced in programs.

To return to the default in which no indicator structure is generated, either remove the relevant SET directive or alter it to read:

```
SET CM_INDICATOROPT :OFF:
```

Refer to "SQL PRODUCE" on page 236 for details of the SQL PRODUCE command.

### **Tailoring DATE and TIME Character Field Lengths**

You can tailor the character field lengths the SQL PRODUCE command will generate for column variables in host language data structures that correspond to a SQL/DS 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 SQL/DS environment.

To tailor character field lengths corresponding to DATE, enter the following SET directive in your SQL/DS profile:

```
SET CM_LOCAL_DATE n
```

where *n* is the character field length to be generated. The minimum value is 10 which is also the default.

To tailor character field lengths corresponding to TIME, enter the following SET directive in your SQL/DS profile:

```
SET CM_LOCAL_TIME n
```

where *n* is the character field length to be generated. The minimum value is 8 which is also the default.

To return to the defaults either remove the relevant SET directives or specify a value of 10 and 8 for *n*.

Refer to "SQL PRODUCE" on page 236 for details of the SQL PRODUCE command.

# 6

## Generation of SQL/DS Application Programs

This chapter will be used to document future enhancements to the support ManagerProducts provides for your SQL/DS environment.

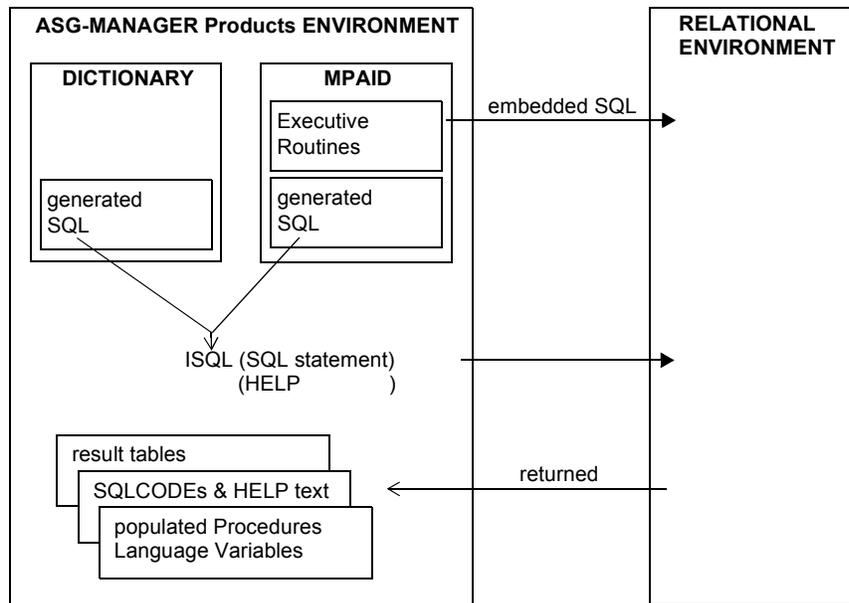


Figure 24 Dynamic SQL Services



---

# 7

## Dynamic SQL Services

---

### Introduction to Dynamic SQL Services

#### Overview of Dynamic SQL Services

The Dynamic SQL Services feature enables you to dynamically submit SQL statements to your DB2 or SQL/DS environment, and receive the results, from within Manager Products.

Dynamic SQL Services are provided by the Translation and Transfer Engine facility.

To use Dynamic SQL Services you must have Manager Products and either DB2 or SQL/DS available on the same CPU and operating system.

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. The SQL statements can be submitted with the ISQL command or embedded in Executive Routines.

Using the Manager Products Procedures Language these Executive Routines can update your DB2 or SQL/DS environment with information held in the dictionary, or, import information from DB2 or SQL/DS into the Manager Products environment.

SQL statements can be submitted both interactively and in batch.

Dynamic SQL Services therefore provide a dynamic link between your DB2 or SQL/DS environment and the integrated Manager Products Family of Program Products.

Refer to "ISQL" on page 149 for details of the ISQL command.

Refer to "Creating Executive Routines to Dynamically Submit SQL Statements to Your DB2 or SQL/DS Environment" on page 109 for details of the Executive Routines with which you can dynamically submit SQL statements.

#### Security and Authorization

The SQL statements you can submit using Dynamic SQL Services are determined by the privileges granted to your *authorization ID* in DB2 or SQL/DS.

Dynamic SQL Services takes your operating system logon in the environment in which DB2 or SQL/DS operates, and not your ASG-ControlManager Logon Identifier, as your authorization ID.

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 Printed by Dynamic SQL Services**

The output printed by Dynamic SQL Services will vary 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 MPDYDSSSEL.

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 `SQLI_ROWS` 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 Systems Administrator can specify the maximum line limit with the `SET OUTPUT-LINE-LIMIT` command. Use the `QUERY OUTPUT-LINE-LIMIT` command to find out the current line limit. The maximum line limit does not affect result tables printed in response to SQL statements submitted in batch.

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 ? symbol 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 MPDYDSSSEL 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.

Refer to the *ASG-Manager Products Systems Administrator's Manual* for details of the SET OUTPUT-LINE-LIMIT command.

Refer to "The COMMAND and EXECUTIVE Members Used in Dynamic SQL Services" on page 115 for details of the COMMAND and EXECUTIVE members provided for use in Dynamic SQL Services.

## **Creating Executive Routines to Dynamically Submit SQL Statements to Your DB2 or SQL/DS Environment**

### ***Introduction to Dynamically Submitting SQL Statements from within Executive Routines***

#### ***Overview of Submitting SQL Statements from within Executive Routines***

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 dictionary 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 or you can apply the tools in the Manager Products Family of Program Products to it. For example, you could 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.

**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 and can contain a maximum of 60,000 separate array elements. Each element represents a value in the column.

For example, three Command Variables named V1, V2, and V3 each having three elements, would represent a table with three columns and rows as follows:

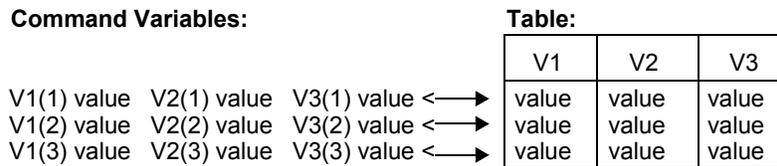


Figure 25 Defining a Table with Command Variables

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) would create a table with three columns and rows as follows:

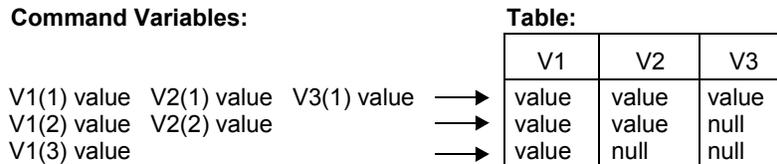


Figure 26 Creating a Table with Command Variables

You can import information about tables and views. 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 about a table with null values would be assigned to the elements of the Command Variables V1, V2, and V3 as follows:

| <b>Command Variables:</b>   | <b>Table:</b>  |       |    |    |       |       |       |       |      |       |       |       |      |
|---|--|-------|----|----|-------|-------|-------|-------|------|-------|-------|-------|------|
| V1(1) value   V2(1) value   V3(1) value <—→<br>V1(2) value   V2(2) null   V3(2) value <—→<br>V1(3) value   V2(3) value   V3(3) null <—→ | <table border="1"> <thead> <tr> <th>V1</th> <th>V2</th> <th>V3</th> </tr> </thead> <tbody> <tr> <td>value</td> <td>value</td> <td>value</td> </tr> <tr> <td>value</td> <td>null</td> <td>value</td> </tr> <tr> <td>value</td> <td>value</td> <td>null</td> </tr> </tbody> </table> | V1    | V2 | V3 | value | value | value | value | null | value | value | value | null |
| V1  | V2   | V3    |    |    |       |       |       |       |      |       |       |       |      |
| value   | value  | value |    |    |       |       |       |       |      |       |       |       |      |
| value   | null   | value |    |    |       |       |       |       |      |       |       |       |      |
| value   | value  | null  |    |    |       |       |       |       |      |       |       |       |      |

Figure 27 Assigning Information about a Table to Command Variables

### 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 Procedures Language values assigned to the different elements of the 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. Any rows following the row with the incorrect value *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. Any rows following the row with the incorrect value *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).

If you want to create or insert rows into tables with columns having data types other than INTEGER or VARCHAR(254) you can use the ISQL command or create an Executive Routine calling the COMMAND member MPDYDSSSQL.

### **Importing Information from Columns with Particular Data Types**

You can create an Executive Routine which imports 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 14 Column Data Types and Procedures Language Values**

| <b>Column Data Type</b> | <b>Procedures Language Value</b>    |
|-------------------------|-------------------------------------|
| 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         | Not supported                       |
| CHAR (n)                | CHARACTER                           |
| VARCHAR (n)             | CHARACTER                           |
| LONG VARCHAR            | CHARACTER                           |
| SMALLINT                | NUMERIC                             |

**Table 14 Column Data Types and Procedures Language Values**

| Column Data Type | Procedures Language Value            |
|------------------|--------------------------------------|
| INTEGER          | NUMERIC                              |
| FLOAT (n)        | NUMERIC (using the DECIMAL function) |
| DECIMAL (n,m)    | NUMERIC                              |

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 in Dynamic SQL Services

### Introduction to the Variables Used in Dynamic SQL Services

*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* are those Command Variables which are specified in Executive Routines. For example, the `SQLI_COMMAND` variable with which you specify the SQL statements to be submitted.

*Return variables* are those Command Variables which are not specified in Executive Routines but are generated by Dynamic SQL Services 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

The `SQLI_COMMAND` Command Variable 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 commence 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 any elements specified in an earlier variable are set to null in subsequent variables in which they are unused. For example, if the first SQL statement you submit uses three elements of the `SQLI_COMMAND` variable but the second SQL statement you submit only uses two elements, you must set the third element to null.

The `SQLI_TABLE_SPACE` Command Variable defines the name of the DB2 table space or SQL/DS dbspace in which a table is stored. The dbspace or table space must already exist.

The `SQLI_TABLE_NAME` Command Variable 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.

The `SQLI_ROWS` Command Variable 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 will override 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**

The `SQLI_CD_n` Command Variable (where *n* is the number of the column) contains the values in the columns from which you have imported information. The variable is generated as a default if you have not named the variables to which the information is to be assigned.

The `SQLI_CS` Command Variable 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.

The `SQLI_CL` Command Variable contains the names of the columns from which you have imported information.

The `SQLI_RETURN` Command Variable contains the Manager Products code returned by Dynamic SQL Services. The return code is the same as the number of the Manager Products message it generates.

The `SQLI_SQLCODE(1)` Command Variable contains the SQL/DS or DB2 SQLCODE returned from your relational environment.

The `SQLI_SQLCODE(2)` Command Variable contains the variable-names within the SQL/DS HELP text associated with the returned SQLCODE.

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.

### **The *COMMAND* and *EXECUTIVE* Members Used in Dynamic SQL Services**

The following *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 *SQL/DS HELP* text

### **Creating and Populating a Table**

You can create an Executive Routine which creates a table and specifies the DB2 table space or SQL/DS dspace 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*
- Those variables defining the columns in the table

and can optionally contain the Command Variable and directives:

- *SQLI\_ROWS*
- Those directives defining the values in the columns.

The called *COMMAND* member *MPDYDSSCRT* must be followed in the Executive Routine by the names, enclosed in literal delimiters, of the columns you have defined.

By specifying *DACCESS* and *DRETRIEVE* commands in the Executive Routine you can populate the table with information filed in members in the dictionary.

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 following Executive Routine:

---

```
MPXX
/*
/* _____
/* An Example of an Executive Routine which creates and
/* populates a table.
/* _____
LITERAL :
/* _____
/* 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
/* _____
/* Calling the COMMAND member MPDYDSSCRT.
/* _____
MPDYDSSCRT :AREA: :CATALOGUE_NAME: :QTY: ;
/* _____
/* Calling the EXECUTIVE member MPDYDSSXIT.
/* _____
MPDYDSSXIT;
/* _____
/* Either exit or create and populate another table.
```

```

/* _____
EXIT

```

Figure 28 Dynamically Creating and Populating a Table

Refer to "Variables Used in Dynamic SQL Services" on page 113 for details of the variables used in Dynamic SQL Services.

### **Inserting Rows into a Table**

You can create an Executive Routine which inserts rows into a table.

The Executive Routine must call:

- The COMMAND member MPDYDSSINS to insert rows into the table
- The EXECUTIVE member MPDYDSSXIT to display any SQLCODEs and HELP text returned from your relational environment

and contain the Command Variables:

- SQLI\_TABLE\_NAME
- SQLI\_TABLE\_SPACE
- Those variables and directives specifying the names of the columns and defining the rows to be inserted

and can optionally contain the Command Variable:

- SQLI\_ROWS.

The called COMMAND member MPDYDSSINS must be followed in the Executive Routine by the names, enclosed in literal delimiters, of the columns in the table into which you are inserting rows.

By specifying DACCESS and DRETRIEVE commands in the Executive Routine you can insert information filed in members in the dictionary into the table.

For example, to insert the third and fourth rows into the following table named SALES:

| AREA  | CATALOGUE_NAME | QTY  |
|-------|----------------|------|
| NORTH | D35            | null |
| SOUTH | D36            | 5500 |
| EAST  | null           | null |
| WEST  | D37            | null |

enter the following Executive Routine:

---

```
MPXX
/*
/* Example Executive Routine to insert rows into a table.
/*
LITERAL :
/*
/* 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 columns names.
/*
COMMAND AREA
COMMAND CATALOGUE_NAME
COMMAND QTY
/*
/* Defining the rows to be inserted.
/*
DACCESS MEMBER :GROUP-SALES-EW:;
DRETRIEVE ALL ALIAS;
DRETRIEVE ALL CATALOGUE;
AREA() = ALIAS_NAME
CATALOGUE_NAME(2) = D37
/*
/* Call COMMAND member MPDYSSINS.
/*
MPDYSSINS :AREA: :CATALOGUE_NAME: :QTY: ;
/*
/* Call EXECUTIVE member MPDYSSXIT.
/*
MPDYSSXIT;
/*
/* Either exit or insert rows into another table.
/*
EXIT
```

---

Figure 29 Dynamically Inserting Rows into a Table

Refer to "Variables Used in Dynamic SQL Services" on page 113 for details of the variables used in Dynamic SQL Services.

### **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 Variables:

- SQLI\_ROWS
- Those naming the 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 names 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 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 and views. 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 following Executive Routine:

---

```
MPXX
/*
/* _____
/* Example Executive Routine for importing information and
/* assigning it to Command Variables.
/* _____
LITERAL :
/* _____
```

```
/* 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.
/* _____
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
/* _____
/* Manipulate the variables and values to display the
/* result table.
/* _____
SAY LEFT (SQLI_CL(1), SQLI_CS(1)) -
LEFT (SQLI_CL(2), SQLI_CS(2)) -
LEFT (SQLI_CL(3), SQLI_CS(3))
I = 1
-BEGLOOP1
IF I > ARRAYHI (:AREA:) THEN GOTO ENDLOOP1
SAY LEFT (AREA(I), SQLI_CS(1)) -
LEFT (CATALOGUE_NAME(I), SQLI_CS(2)) -
LEFT (QTY(I), SQLI_CS(3))
I = I+1
GOTO BEGLOOP1
-ENDLOOP1
EXIT
```

---

Figure 30 Dynamically Importing Information

Refer to "Variables Used in Dynamic SQL Services" on page 113 for details of the variables used in Dynamic SQL Services.

### **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 following Executive Routine:

---

```
MPXX
/* _____
/* Example Executive Routine to generate any SQL statement
/* that can be prepared.
/* _____
LITERAL :
/* _____
/* Specifying the SQL statement.
/* _____
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
```

---

Figure 31 Dynamically Submitting SQL Statements

Refer to "Variables Used in Dynamic SQL Services" on page 113 for details of the variables used in Dynamic SQL Services.

## 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.

# 8

## Import

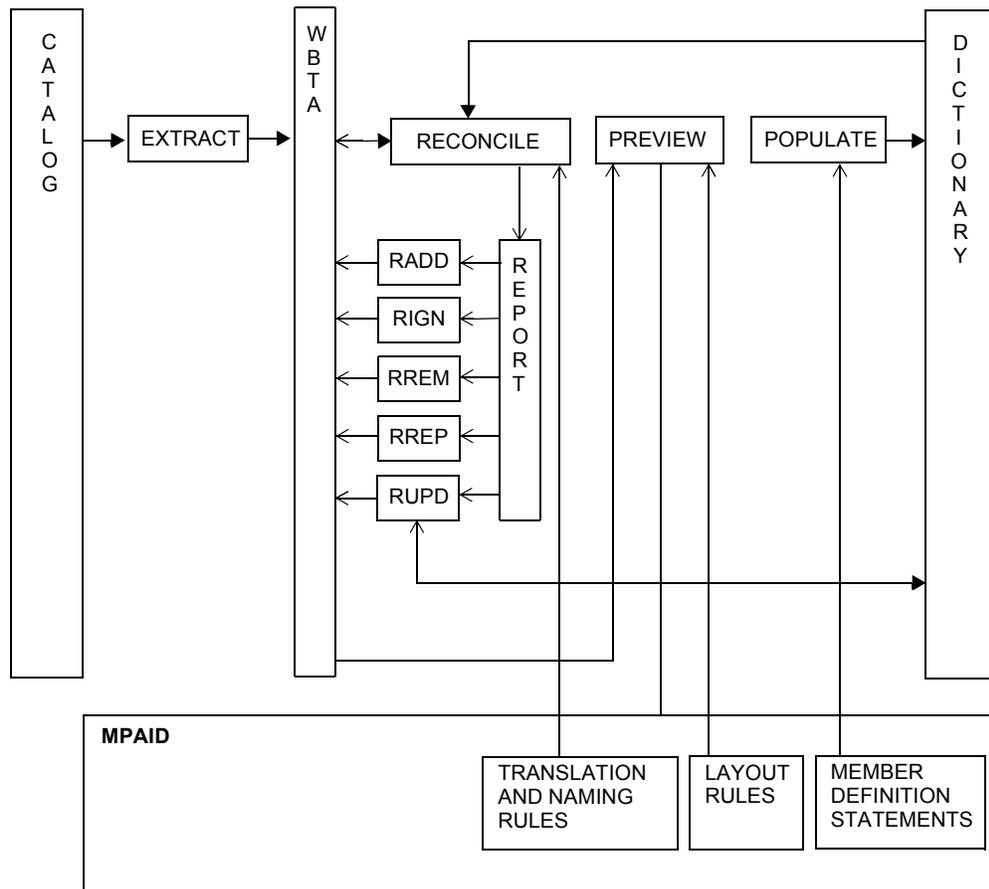


Figure 32 An Illustration of How Information is Imported

## Introduction to Importing Information about External Objects

### Overview of Importing Information

The Corporate Dictionary/Repository Definition Import from SQL/DS facility enables you to import information about SQL/DS objects from the SQL/DS catalog into the Manager Products environment.

Use the EXTRACT SQL command to import information. The EXTRACT SQL command passes SQL statements to SQL/DS which interrogate the catalog. The results of the interrogations are used to populate the WorkBench Translation Area (WBTA) with Manager Products Procedures Language Global Variables. The information on the WBTA can be subsequently used by the RECONCILE, PREVIEW, and POPULATE commands to generate proposed members which document SQL/DS objects and to enter the members in the dictionary.

To be able to import information with the EXTRACT SQL command you must have SQL/DS on the same CPU and running under the same operating system as Manager Products.

Access to the information in the catalog is controlled by SQL/DS. The information you can import is determined by the privileges granted to SQL/DS authorization IDs. Your ASG-ControlManager Logon Identifier is taken as your authorization ID unless you have specified another authorization ID in the CREATOR clause of the EXTRACT SQL command.

The members documenting the SQL/DS objects are of the member types provided by the SQL/DS Definition Facility. For example, information about a table is documented in a SQL-TABLE member. Imported information about columns and their data type is documented in ITEM members.

The members are given default names which include a prefix identifying the type of object they document and, for some types of object, the authorization ID of the owner of the object in SQL/DS.

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 dictionary standards.

Importing information about an object will result in information about particular objects directly related to it also being imported. The proposed members documenting the objects include in their definitions, clauses which define their relationships to one another.

Manager Products users who have not already documented their SQL/DS environment in the dictionary can therefore do so in the minimum amount of time. Users who have already documented their SQL/DS environment in the dictionary can reconcile their existing member definitions with the objects in SQL/DS so as to ensure that their documentation is both accurate and complete.

Having documented your SQL/DS environment in the dictionary you can use the tools in the integrated Manager Products Family of products to analyze, enhance, and maintain that environment.

## **Naming Guidelines When Importing Information**

The information imported about objects in your relational environment can be used to generate proposed members. Different types of object are documented in particular types of member.

Default 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, and
- For certain types of objects, the authorization ID of the owner of the object in your relational environment, and
- The name of the object on the catalog

with each part of the member name being separated by a hyphen.

For example, a proposed member documenting a relational table is named TB-owner-name.

The name of the objects as held on the catalog (excluding the authorization ID) is entered in the ALIAS clause of proposed members. The names of columns as held on the catalog are entered in the KNOWN-AS clauses of proposed members documenting tables.

Underscores in the names of external objects are converted into hyphens when information is imported about the object.

You can rename proposed members using the RECONCILE and RREN commands.

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 dictionary. To find out which characters can be used to delimit member names use the QUERY STRING-DELIMITER command.

If you have already documented your relational environment in the dictionary, 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 dictionary members if the proposed member's names do not comply with your naming standards.

Reconciliation is carried out by the RECONCILE command which displays a report comparing the proposed members with any existing dictionary members which have the same name.

An inaccurate reconciliation can result in objects in your relational environment being documented in more than one dictionary member and the one to one correspondence between external objects and dictionary members being lost.

If you have already documented your relational environment in the dictionary you should take into consideration the following when importing information:

- A column *can be* documented in a GROUP member but imported information about a column is always documented in an ITEM member. The ITEM members have their own default naming prefix and may not be reconciled with any existing GROUP members documenting the same columns.

When importing information about external objects which were originally created by exporting from member definitions in the dictionary, you should take into consideration the following:

- An object name generated from the KNOWN-AS or ALIAS clause of a member is not reconciled with the name of the member whose definition contains the clause
- An object name generated from a member name longer than that allowed in your relational environment is reduced in length by the *Name Reduction Process*. The reduced object name is not reconciled with the full member name.

The Systems Administrator can define alternative naming rules which suit your naming standards, for example, to create your own naming prefixes. Alternatively the interrogation of the dictionary carried out during reconciliation can be tailored.

For example if it is your naming standard to generate object names from KNOWN-AS or ALIAS clauses you could specify WHOSE ALIAS IS or WHICH MEMBERS HAVE KNOWN-AS EQUALS commands during reconciliation so as to find out if there is an existing dictionary member documenting an object. This approach is only effective if you have, as a dictionary standard, entered unique names in all ALIAS or KNOWN-AS clauses.

You could also specify the full name of dictionary members (if they do not exceed thirty characters) as the labels of external objects. Labels are not reduced in length by the Name Reduction Process and any hyphens within them are not changed to underscores when SQL statements are generated from a member's definition. When importing information about the objects you could reconcile the object's label with the member's name.

Refer to "Deriving SQL/DS Object Names and External Names from Aliases" on page 101 for details of how to define your own naming rules.

Refer to the Appendix, "The Manager Products Name Reduction Process," on page 279 for details of the Name Reduction Process.

Refer to the *ASG-ControlManager User's Guide* for details of the Manager Products character set.

**Table 15 Naming Rules for Members Documenting Imported Objects**

|         |             |                       |
|---------|-------------|-----------------------|
| COLUMN  | ITEM        | IT- <i>name</i>       |
| DBSPACE | SQL-DBSPACE | SD- <i>name</i>       |
| INDEX   | SQL-INDEX   | SX- <i>owner-name</i> |
| OWNER   | SQL-USER    | SR- <i>name</i>       |
| TABLE   | SQL-TABLE   | ST- <i>owner-name</i> |
| VIEW    | SQL-VIEW    | SV- <i>owner-name</i> |

where *name* is the name of the object on the SQL/DS catalog and *owner* is the SQL/DS authorization ID of the owner of the object.

## **How Columns Are Documented**

Imported information about columns in your relational environment is documented in ITEM members.

The data type of a column is documented in the form-description and USAGE clause of the ITEM member. The ITEM members are given a form-keyword of HELD-AS.

Columns can be documented 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 dictionary.

Refer to "Description of MPDYWTCVDT" on page 130 for details of how to tailor how columns are documented.

The following table shows how the data type of a column is documented in the form-description and USAGE clause of ITEM members.

**Table 16 Documenting the Data Type of Imported Columns**

| Column Data Type                       | Form-Description                | USAGE Clause |
|--|---------------------------------|--------------|
| TIME                                   | CHARACTER 8                     | TIME         |
| DATE                                   | CHARACTER 10                    | DATE         |
| TIMESTAMP                              | CHARACTER 26                    | TIMESTAMP    |
| GRAPHIC( <i>n</i> )                    | CHARACTER <i>n</i>              | GRAPHIC      |
| VARGRAPHIC( <i>n</i> )                 | CHARACTER 1 TO <i>n</i>         | GRAPHIC      |
| LONG VARGRAPHIC                        | CHARACTER 16383                 | GRAPHIC      |
| SMALLINT                               | BINARY 4                        | none         |
| INTEGER                                | BINARY 9                        | none         |
| FLOAT( <i>n</i> )<br><i>n</i> = 1..21  | FLOATING-POINT 6                | none         |
| FLOAT( <i>n</i> )<br><i>n</i> = 22..53 | FLOATING-POINT 16               | none         |
| DECIMAL( <i>n,m</i> )                  | PACKED-DECIMAL ( <i>n-m,m</i> ) | none         |
| CHARACTER( <i>n</i> )                  | CHARACTER <i>n</i>              | none         |
| VARCHAR( <i>n</i> )                    | CHARACTER 1 TO <i>n</i>         | none         |
| LONG VARCHAR                           | CHARACTER 32767                 | none         |

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 clause as PACKED-DECIMAL 2.3.

## Tailoring Import Commands

### Introduction to Tailoring Import Commands

We provide predefined facilities for importing information from DB2 and SQL/DS. There are four stages to the import procedure:

- Extract (using the EXTRACT DB2 and EXTRACT SQL commands)
- Reconcile (using the RECONCILE command)
- Preview (using the PREVIEW IMPORT command)
- Populate (using the POPULATE command).

You can tailor two of these stages (the RECONCILE and PREVIEW IMPORT commands), in order to customize imported information to suit your own purposes and environment.

The EXTRACT DB2 and EXTRACT SQL commands enable you to extract information from the DB2 and SQL/DS Catalogs respectively, and place it on the Workbench Translation Area (WBTA) in the form of Global Variables containing proposed member names and member types. These commands are provided by the Corporate Dictionary/Repository Definition Import from DB2 facility, and the Corporate Dictionary/Repository Definition Import from SQL/DS facility respectively.

You can then reconcile this extracted information against dictionary information, using the RECONCILE command, and generate member definition statements from it, using the PREVIEW IMPORT command. These commands are provided by the Transfer and Translation Engine facility.

**Note:** \_\_\_\_\_

You can tailor some aspects of the extraction process at installation time. Refer to the information about Manager Products and external environments in your Manager Products installation manual.

You can tailor the RECONCILE command to change the way in which it reconciles extracted information with information already present in your dictionary.

You can tailor the PREVIEW IMPORT command to generate member definition statements in a format that suits your environment, for example by omitting the ALIAS clause from member definitions.

Thus, you can customize the reconciliation procedure and member generation to suit your installation's requirements.

Both the RECONCILE and PREVIEW IMPORT commands call a series of Corporate Executive Routines in the course of execution. In order to tailor either command you must change the instructions in Corporate Executive Routines.

The RECONCILE command calls the following Corporate Executive Routines:

- MPDYWTDFLT, to:
  - Optionally call MPDYWTMT42
  - Set up default names for these Corporate Executive Routines:
- MPDYWTCVDT, to convert SQL column data types
- MPDYWTRDMR, to generate proposed member names and proposed member types from extracted object information
- MPDYWTOCOD, to convert SQL keyword codes to Manager Products keywords
- MPDYWTEXCC, to extract common clauses from dictionary
- MPDYWTMT42, to establish member type checking for DB2 and SQL/DS members.

You can also arrange for RECONCILE to call your own Executive Routines, by specifying their names in the USING clause of the RECONCILE command. They will then be executed after MPDYWTOCOD and before MPDYWTEXCC.

The PREVIEW IMPORT command calls the following Corporate Executive Routines:

- MPDYMMCNTL, to control which Corporate Executive Routine for layout rules is executed, depending on the proposed member type
- Corporate Executive Routines for DB2 member types
- Corporate Executive Routines for SQL/DS member types
- MPDYMMLOCC, to set up common clauses
- MPDYMMLOIT, to set up ITEM definitions.

If you want to obtain a print of any of the Corporate Executive Routines mentioned above, enter:

```
MP PRINT EXECUTIVE corporate-executive-name ;
```

The Corporate Executive Routines are supplied to you as Executive Routine members in the Manager Products Administration dictionary. They must be constructed onto the MP-AID before use.

A possible method of tailoring a Corporate Executive Routine, provided you have the User Defined Commands facility installed, is to copy it to a USER-MEMBER with the same name. You can then tailor the USER-MEMBER, leaving the master copy intact. The USER-MEMBER will be called in preference to the Corporate Executive Routine the next time you run the command.

Refer to "Member Type and Command Descriptions" on page 143 for details of the RECONCILE command and the PREVIEW IMPORT command.

## Tailorable Corporate Executive Routines

### Corporate Executive Routines for SQL/DS Dictionary Definitions

The Corporate Executive Routine MPDYMMCNTL calls the Corporate Executive Routines listed below to generate definitions for proposed dictionary members. Each corporate Executive Routine generates a dictionary definition for a particular member type.

The following table shows each SQL/DS object for which you can generate a dictionary definition, the Corporate Executive Routine which generates it, and the generated dictionary member type.

| SQL/DS Object    | Corporate Executive Routine | Dictionary Member |
|------------------|-----------------------------|-------------------|
| column           | MPDYMMLOIT                  | ITEM              |
| authorization-ID | MPDYMMLOUS                  | SQL-USER          |
| table            | MPDYMM32TB                  | SQL-TABLE         |
| view             | MPDYMM32VW                  | SQL-VIEW          |
| dbspace          | MPDYMM32DB                  | SQL-DBSPACE       |
| index            | MPDYMM32 IN                 | SQL-INDEX         |

These Corporate Executive Routines all do the following:

- They generate REPLACE or ADD statements, followed by the appropriate data definition for the proposed member
- They call the Corporate Executive Routine MPDYMMLOCC to set up common clauses for the proposed member.

### Description of MPDYWTDFLT

This Corporate Executive Routine sets up the default initialization Executive Routines which are called by the RECONCILE command, and the default form of the CONTAINS clause for DB2-TABLE and DB2-VIEW member definitions.

It also optionally executes the supplied executive MPDYWTMT42 to establish member type checking during reconciliation for DB2 and SQL/DS.

### Description of MPDYWTCVDT

This Corporate Executive Routine converts SQL data types for columns into ITEM form descriptions. It converts each column data type, setting up a dmr\_mem\_desc variable according to default conversion rules which conform to those given in the description of how to document SQL/DS and DB2 column data types.

### Description of MPDYWTRDMR

This Corporate Executive Routine contains the rules for converting SQL/DS and DB2 object names and types into proposed dictionary member names and member types.

### *Description of MPDYWTOCOD*

This Corporate Executive Routine converts SQL keyword codes, which represent attributes specific to the SQL/DS and DB2 objects which can be imported, into dictionary member codes. For example, 'U', which represents a Unique Index, is converted to 'UNIQUE'.

### *Description of MPDYWTEXCC*

This Corporate Executive Routine extracts common clauses from the dictionary. It is passed 2 parameters:

- &p0: Is the dmr\_mem\_name array number of the current member being processed.
- &pl: Is the dmr\_mem\_name array number of the first member in a chain of members with the same name. This value will be the same as &p0 if the current member is being processed for the first (or only) time. If the two values differ, then the current member (at &p0) has already been processed (at &pl) and the common clause variables associated with &pl can be used for &p0; that is, no more DRETRIEVES are required.

### *Description of MPDYWTMT42*

This Corporate Executive Routine sets up member type checking for DB2 and SQL/DS member types during the RECONCILE stage.

### *Description of MPDYMMLOCC*

This Corporate Executive Routine sets up the common clauses for proposed dictionary members generated by the PREVIEW IMPORT command.

## ***Global Variables Used in the Import Commands***

### *Introduction to the Variables Used in the Import Commands*

The RECONCILE and PREVIEW IMPORT commands use global variables to manipulate and store information extracted from the DB2 and SQL/DS Catalogs. The global variables are grouped according to how they are used, as shown below.

Variables specific to SQL:

- DBSPACE data variables (EXT\_DBS\_).

Variables for DB2 and SQL/DS:

- 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 variables:

- Generic import variables (EXT\_OBJ\_)
- Proposed dictionary member variables (DMR\_MEM\_)
- Variables for existing dictionary members (DMR\_DIC\_)
- Common clause variables (DMR\_DIC\_)
- References from existing dictionary members (DMR\_REF\_).

### **DBSPACE Data Variables**

Variables which have names beginning EXT\_DBS refer to extracted DBSPACE data.

| <b>Variable Name</b> | <b>Source (SQL/DS)</b>   |
|----------------------|--|
| EXT_DBS_LOCK         | sysdbspaces.lockmode   |
| EXT_DBS_NHEADER      | sysdbspaces.nrheader   |
| EXT_DBS_OWNER_PTR    | pointer to the EXT_OBJ_NAME array for the creator/owner of the DBSPACE |
| EXT_DBS_PAGES        | sysdbspaces.npages   |
| EXT_DBS_PCTFREE      | sysdbspaces.freepct  |
| EXT_DBS_PCTINDEX     | sysdbspaces.pctindx  |
| EXT_DBS_STORPOOL     | sysdbspaces.pool   |
| EXT_DBS_TYPE         | sysdbspaces.dbspacetype  |

The table above lists each available DBSPACE data variable, with one of the following:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)
- The source of the extracted value in the SQL/DS Catalog.

Refer to the *IBM SQL/DS Reference Manual* for the exact meaning and use of the extracted values.

### **Column Data Variables**

Variables which have names beginning EXT\_COL refer to column data.

| <b>Variable Name</b> | <b>Source (DB2)/Use</b>  | <b>Source (SQL/DS)</b> |
|----------------------|--|------------------------|
| EXT_COL_BIT_DATA     | syscolumns.foreignkey  | syscolumns.foreignkey  |
| EXT_COL_COMMENT      | syscolumns.remarks   | 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   | syscolumns.tcreator    |
| EXT_COL_DEFAULT      | syscolumns.default   | syscolumns.default     |

| Variable Name       | Source (DB2)/Use  | Source (SQL/DS)   |
|---------------------|---|---|
| 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  | 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 clause in a VIEW; otherwise it contains null (VIEWS ONLY).                  |   |
| EXT_COL_LABEL       | syscolumns.label  | syscolumns.label  |
| EXT_COL_LENGTH      | syscolumns.length   | syscolumns.syslength (= x in EXT_COL_PRECISION and EXT_COL_SCALE) |
| EXT_COL_NAME        | Contains the name of the column.  |   |
| EXT_COL_NULLS       | syscolumns.nulls  | syscolumns.nulls  |
| EXT_COL_NUMBER      | syscolumns.colno  | syscolumns.colno  |
| EXT_COL_PKEY        | Contains 'PRIMARY-KEY' if the column is a primary key.  |   |
| EXT_COL_PKEY_SEQ    | SQL/DS only. The sequence of a column in a key:<br>ASC = ascending<br>DSC = descending  |   |
| EXT_COL_PSEQUENCE   | syscolumns.keyseq   | syscolumns.keyseq   |
| EXT_COL_PRECISION   | syscolumns.(length- scale)  | $(x/256)-(x-(x/256)*256)$   |
| EXT_COL_SCALE       | syscolumns.keyseq   | $(x-(x/256)*256)$   |
| EXT_COL_SEQUENCE    | syskeys.ordering  |   |
| EXT_COL_TNAME       | syscolumns.tbname   | syscolumns.tname  |
| EXT_COL_TYPE        | syscolumns.coltype  | syscolumns.coltype  |

The table lists each available column data variable, with either:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)

or:

- The source of the extracted value in the DB2 Catalog (column 2)
- The source of the extracted value in the SQL/DS Catalog (column 3).

Refer to the IBM DB2 and SQL/DS reference manuals for the exact meaning and use of the extracted values.

**INDEX Data Variables**

Variables which have names beginning EXT\_IND refer to INDEX data.

| <b>Variable Name</b>  | <b>Source (DB2)/Use</b>  | <b>Source (SQL/DS)</b> |
|-----------------------|--|------------------------|
| EXT_IND_BUFPOOL       | sys indexes.bpool  |                        |
| EXT_IND_CLOSE         | sysindexes.closerule   |                        |
| EXT_IND_CLUSTER       | sysindexes.clustering  |                        |
| EXT_IND_CNOS          |  | sysindexes.colnumbers  |
| 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    | Indicates the sequence in which the column entries are indexed. It may be ASCENDING or DESCENDING. |                        |
| EXT_IND_CREATOR       | sysindexes.creator   | sysindexes.icreator    |
| 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_FREEPAGE      | sysindexpart.freepage  |                        |
| EXT_IND_PCTFREE       | sysindexpart.pctfree   | sysindexes.ipctfree    |
| EXT_IND_PRIQTY        | sys indexpart.pqty   |                        |
| EXT_IND_SECQTY        | sysindexpart.secqty  |                        |
| EXT_IND_STORGROUP_PTR | Pointer to the EXT_OBJ_NAME array that names the storage group used for the index.                 |                        |
| EXT_IND_STORNAME      | sysindexpart.storname  |                        |
| EXT_IND_SUBPAGE       | Contains the number of subpages per page.  |                        |
| EXT_IND_SUBPAGE_SIZE  | sys indexes.pgsize   |                        |
| EXT_IND_TABLE_PTR     | Pointer to the EXT_OBJ_NAME array that names the indexed table.                                    |                        |
| EXT_IND_TCREATCR      | sysindexes.tbcreator   | sysindexes.creator     |
| EXT_IND_TNAME         | sysindexes.tbname  | sysindexes.tname       |
| EXT_IND_TYPE          | sysindexes.uniquerule  | sysindexes.indextype   |
| EXT_IND_VCATNAME      | sysindexpart.vcatname  |                        |

The table above lists each available INDEX data variable, with either:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)

or:

- The source of the extracted value in the DB2 Catalog (column 2)
- The source of the extracted value in the SQL/DS Catalog (column 3).

Refer to the IBM DB2 and SQL/DS reference manuals for the exact meaning and use of the extracted values.

### *TABLE Data Variables*

Variables which have names beginning EXT\_TAB refer to extracted TABLE data.

| <b>Variable Name</b>  | <b>Source (DB2)/Use</b>  | <b>Source (SQL/DS)</b>   |
|-----------------------|--|--|
| EXT_TAB_AUDIT         | systables.auditing   | systables.auditing   |
| EXT_TAB_COL_OCC       | systables.colcount   | syscatalog.ncols   |
| 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  | syscatalog.remarks   |
| EXT_TAB_CORREL        | Identifier that designates the table or view.  |  |
| EXT_TAB_CREATOR       | systables.creator  | syscatalog.creator   |
| EXT_TAB_CREATOR_PTR   | Pointer to the EXT_OBJ_NAME array that names the creator of the table.                           |  |
| EXT_TAB_DATABASE      | 'DB2 V.1' for DB2 Version 1 Release 3, 'DB2 V.2' for DB2 Version 2 Release 2                     | 'SQL/DS V.1' for SQL/DS Version 2 Release 1, 'SQL/DS V.2' for SQL/DS Version 2 Release 2 |
| EXT_TAB_EDPROC        | systables.edproc   | systables.edproc   |
| EXT_TAB_EDPROC_PTR    | Pointer to the EXT_OBJ_NAME array for the edit procedure.  |  |
| EXT_TAB_FKEY_CCREATOR | sysforeignkeys.creator   | syskeyscols.tcreator   |
| EXT_TAB_FKEY_CNAME    | sysforeignkeys.colname   | syskeycols.cname   |
| EXT_TAB_FKEY_CTNAME   | sysforeignkeys.tbname  | syskeyscols.tname  |
| EXT_TAB_FKEY_DELRULE  | sysrels.deleterule   | syskeys.deleterule   |
| EXT_TAB_FKEY_NAME     | sysforeignkeys.relname   | syskeycols.keyname   |
| EXT_TAB_FKEY_OCC      | systables.parents  | syscatalog.parents   |
| EXT_TAB_FKEY_PTR      | Pointer to the EXT_OBJ_NAME array for the first foreign key.                                     |  |

| <b>Variable Name</b>     | <b>Source (DB2)/Use</b>  | <b>Source (SQL/DS)</b> |
|--------------------------|--|------------------------|
| EXT_TAB_LABEL            | systables.label  | syscatalog.tlabel      |
| EXT_TAB_COL_FCOL_PTR     | Pointer to the EXT_OBJ_NAME array for the foreign key which corresponds to the primary key.                      |                        |
| 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   | syscatalog.rowcount    |
| EXT_TAB_SPACE            | systables.tsname   | syscatalog.dbspacename |
| 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   | sysdbspaces.owner      |
| EXT_TAB_VALPROC          | systables.valproc  |                        |
| EXT_TAB_VALPROC_PTR      | Pointer to the EXT_OBJ_NAME array for the validation procedure.  |                        |

The table above lists each available TABLE data variable, with either:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)

or:

- The source of the extracted value in the DB2 Catalog (column 2)
- The source of the extracted value in the SQL/DS Catalog (column 3).

Refer to the IBM DB2 and SQL/DS reference manuals for the exact meaning and use of the extracted values.

### **VIEW Data Variables**

Variables which have names beginning EXT\_VIE and EXT\_B refer to extracted VIEW data.

| <b>Variable Name</b> | <b>Source (DB2)/Use</b>   | <b>Source (SQL/DS)</b> |
|----------------------|---|------------------------|
| EXT_BCREATOR         | sysviewdep.bcreator   | sysusage.bcreator      |
| EXT_BNAME            | sysviewdep.bname  | sysusage.bname         |
| EXT_BTYPE            | sysviewdep.btype  | sysusage.btype         |
| 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.   |                        |

| <b>Variable Name</b> | <b>Source (DB2)/Use</b>   | <b>Source (SQL/DS)</b> |
|----------------------|---|------------------------|
| EXT_VIE_COMMENT      | Contains information about the view, supplied by a user via COMMENT command.  |                        |
| EXT_VIE_HAVING_OCC   | Contains the number of array elements which are named in the HAVING clause 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 clause.   |                        |
| EXT_VIE_HAVING       | Array which contains all the extracted HAVING clauses.  |                        |
| EXT_VIE_LABEL        | The label of the VIEW as given by a LABEL ON statement.   |                        |
| EXT_VIE_SELECT_TYPE  | Indicates 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_OCC    | Contains the number of array elements which are named in the WHERE clause 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 clause.   |                        |
| EXT_VIE_WHERE        | Array which contains all the extracted WHERE clauses.   |                        |
| EXT_VIEW_TEXT        | sysviews.text   | sysviews.viewtext      |

The table above lists each available VIEW data variable, with either:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)

or:

- The source of the extracted value in the DB2 Catalog (column 2)
- The source of the extracted value in the SQL/DS Catalog (column 3).

Refer to the IBM DB2 and SQL/DS reference manuals for the exact meaning and use of the extracted values.

**Generic Import Variables**

These variables are generic to the import facility. They are used by the RECONCILE command to generate proposed member names and types. Examples given are from DB2, but they are also used for import from SQL/DS.

| <b>Variable Name</b>   | <b>Source (DB2)/Use</b>   | <b>Source (SQL/DS)</b>  |
|------------------------|---|-------------------------|
| 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  | syscatalog.tname        |
|                        | systcolumns.name  | systcolumns.cname       |
|                        | systdatabase.name   | sysindexes.iname        |
|                        | sysplan.name  | sysdbspaces.dbspacename |
|                        | sysdbrm.name  |                         |
|                        | sysstogroup.name  |                         |
|                        | systablespace.name  |                         |
|                        | systablepart.name   |                         |
|                        | sysindexes.name   |                         |
|                        | syskeys.name  |                         |
| EXT OBJ_CCC            | 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.  |                         |
| EXT OBJ_TYPE           | One of the following literal values, depending on the object being extracted  |                         |
|                        | 'TABLE' or 'VIEW'   | 'TABLE' or 'VIEW'       |
|                        | 'COLUMN'  | 'COLUMN'                |
|                        | 'DATABASE'  | 'INDEX'                 |
|                        | 'PLAN'  | 'DBSPACE'               |

| Variable Name | Source (DB2)/Use  | Source (SQL/DS) |
|---------------|---|-----------------|
|               | '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). |                 |

The table above lists each available generic import variable, with either:

- An explanation of how the variable is used in the Corporate Executive Routines (column 2)
- or:
- The source of the extracted value in the DB2 Catalog (column 2)
  - The source of the extracted value in the SQL/DS Catalog (column 3)
  - The literal value assigned to it by the extract command.

Refer to the *IBM DB2 Reference Manual* for the exact meaning and use of the extracted values.

### Proposed Dictionary Member Variables

Variables with names beginning DMR\_MEM refer to proposed dictionary members. These variables are generated by the RECONCILE command and describe the proposed member names and types.

| Variable Name      | Description  |
|--------------------|--|
| 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 dictionary data type for the object; for example, DB2-TABLE or ITEM.  |
| DMR_MEM_FUNC       | Defines the dictionary update function to be applied to the object. It may contain REPLACE, ADD, or IGNORE.  |
| 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 the proposed member's form description for an ITEM.   |
| DMR_MEM_VERSION    | Contains the proposed version number for a generated ITEM. If no versions exist on the dictionary this will be a null.                                   |

| Variable Name | Description   |
|---------------|---|
| DMR_MEM_GEN   | Indicator used by the PREVIEW IMPORT command, indicating whether or not to generate the definition for the object. It may contain one of the following:<br><br>GEN - generate the definition<br><br>NOGEN - do not generate the definition because:<br><br>a) The parent of this object has DMR_MEM_FUNC set to IGNORE, or<br><br>b) The definition will be generated via another object in this array because another parent refers to the same member.<br><br>REF - do not generate the definition because the member is a REFERENCED OBJECT. |

The table above lists each available dictionary member variable, with an explanation of how the variable is used in the Corporate Executive Routines.

### *Variables for Existing Dictionary Members*

Variables with names beginning DMR\_DIC refer to dictionary members. The following variables are populated when a proposed member already exists on the dictionary.

| Variable Name | Description  |
|---------------|--|
| DMR_DIC_COND  | Defines the condition of the member in the dictionary, in the same format as the CONDITION column seen after a LIST command; for example, SCE ENC, *SCE DUM, or *NC AUTH (that is, member is protected and user has no access authority).  |
| DMR_DIC_MATCH | Indicates whether a proposed member's form description matches a form description for a version of that member in the dictionary. If there is no match, the indicator is set to zero. If there is a match, the indicator indicates the character position in the dictionary form description at which the match begins. For example, if the dictionary form description contains<br>'HELD-AS CHARACTERS 8'<br>and the proposed form description is<br>CHARACTERS 8<br>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 member on the dictionary.  |

| Variable Name    | Description  |
|------------------|--|
| DMR_DIC_VER_OCC  | Contains the number of versions, if the member is held on the dictionary 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 dictionary.  |
| DMR_DIC_VER_PTR  | Contains a pointer to the DMR_DIC_VER_FORM array, if the member is held on the dictionary as an ITEM.                            |

The table above lists each available dictionary member variable, with an explanation of how the variable is used in the Corporate Executive Routines.

### Common Clause Variables

Common clause variables will only be set up if you do NOT specify NO-COMMON-CLAUSES in the RECONCILE command. The number of variables to be set up depends on whether or not a definition already exists in the dictionary for the proposed member.

If the proposed member is not in the dictionary, only the NOTE clause will be set up. It will contain the date and a time stamp.

If the proposed member is already in the dictionary:

- The NOTE clause is created, or updated to include a message and latest time stamp
- Other clauses, if they exist in the dictionary member, are set up as indicated below.

| Variable Name    | Description  |
|------------------|--|
| DMR_DIC_ADM_OCC  | Contains the number of lines of ADMINISTRATIVE-DATA held on the dictionary.                            |
| DMR_DIC_ADM_TEXT | Contains the ADMINISTRATIVE-DATA text for the member.  |
| DMR_DIC_ADM_PTR  | Pointer to the DMR_DIC_ADM_TEXT array which that contains the ADMINISTRATIVE-DATA text for the member. |
| DMR_DIC_ALI_OCC  | Contains the number of ALIASES for the dictionary.   |
| 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_ALI_NAME array that contains the first ALIAS for the member.                    |
| DMR_DIC_CAT_OCC  | Contains the number of lines of CATALOG data held on the dictionary.                                   |
| DMR_DIC_CAT_TEXT | Contains the CATALOG data for the member.  |
| DMR_DIC_CAT_PTR  | Pointer to the DMR_DIC_CAT_TEXT array that contains the CATALOG data for the member.                   |

| <b>Variable Name</b> | <b>Description</b>   |
|----------------------|--|
| DMR_DIC_COM_OCC      | Contains the number of lines of COMMENT data held on the dictionary.                     |
| DMR_DIC_COM_TEXT     | Contains the COMMENT data for the member.  |
| DMR_DIC_COM_PTR      | Pointer to the DMR_DIC_COM_TEXT array that contains the COMMENT data for the member.     |
| DMR_DIC_DES_OCC      | Contains the number of lines of DESCRIPTION data held on the dictionary.                 |
| DMR_DIC_DES_TEXT     | Contains the DESCRIPTION data for the member.  |
| DMR_DIC_DES_PTR      | Pointer to the DMR_DIC_DES_TEXT array that contains the DESCRIPTION data for the member. |
| DMR_DIC_NOT_OCC      | Contains the number of lines of NOTE data held on the dictionary.                        |
| DMR_DIC_NOT_TEXT     | Contains the NOTE data for the member.   |
| DMR_DIC_NOT_PTR      | Pointer to the DMR_DIC_NOT_TEXT array that contains the NOTE data for the member.        |

The table above lists each available common clause variable, with an explanation of how the variable is used in the Corporate Executive Routines.

### **References from Existing Dictionary Members**

If a proposed member already exists on the dictionary and the dictionary member refers to other members, the following DMR\_REF variables will be set up.

| <b>Variable Name</b> | <b>Description</b>  |
|----------------------|---|
| 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.                 |
| DNR_REF_RELATIONSHIP | Contains the relationship between the referenced member and a parent. |

The table above lists each available reference variable, with an explanation of how the variable is used in the Corporate Executive Routines.

---

# 9

## Member Types and Commands

---

### Member Type and Command Descriptions

This section describes the member types and commands provided by Manager Products to support your SQL/DS environment. The member types and commands are documented in alphabetic order of member type name and command name.

#### **EXTRACT SQL**

Use the EXTRACT SQL command to import information about SQL/DS objects onto the Manager Products WBTA.

Refer to "Syntax of the EXTRACT Command" on page 147 for the syntax of the EXTRACT command.

#### *Introduction to the EXTRACT SQL Command*

Use the EXTRACT SQL command to import information about SQL/DS objects into the Manager Products environment.

The EXTRACT SQL command passes SQL statements to SQL/DS which interrogates the SQL/DS catalog. The result tables returned from these interrogations are used to populate the WBTA with Procedures Language Global Variables.

You can import information about the following types of object:

- Columns
- Owners
- Dbspaces
- Tables
- Indexes
- Views.

You can select the objects you wish to import information about by their name and owner. You can also select tables which are stored in specified dbspaces.

Importing information about an object will result in information on particular other objects directly related to it also being imported.

The imported information can be used to generate dictionary members which document the SQL/DS objects. The member's definitions will include clauses 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 dictionary as a dummy member.

### **Importing Information about SQL/DS Objects**

You can import information about a selection of SQL/DS objects into the Manager Products environment.

To import information about SQL/DS objects, enter:

```
EXTRACT SQL object-type object-name-selection ;
```

where:

*object-type* is one of the keywords DBSPACE, INDEX, TABLE, or VIEW and identifies the type of object.

*object-name-selection* is either:

- The name of one or more objects separated by commas, or
- A combination of characters and ? and \* symbols (each symbol representing one or a string of characters) which together match the names of a selection of objects

and identifies the names of the objects.

The object name must be the name (excluding any owner qualifier) of the object as it is known on the SQL/DS catalog.

If your ASG-ControlManager Logon Identifier is different to the authorization ID of the owner of the SQL/DS object you can only import information about it if you specify the correct authorization ID in the CREATOR clause of the EXTRACT SQL command.

You can import information about SQL/DS tables stored in specified dbspaces. This can be useful when you are selecting tables using the ? and \* symbols and want to limit the selection to those tables stored in particular dbspaces.

To import information about SQL/DS tables stored in particular dbspaces, enter:

```
EXTRACT SQL TABLE object-name-selection  
DBSPACE object-name-selection ;
```

### **Importing Information about Objects Owned by any Owner**

You can import information about objects owned by any owner by specifying the owner's SQL/DS authorization ID.

To import information about SQL/DS objects owned by particular owners, enter:

```
EXTRACT SQL object-type object-name-selection  
                CREATOR owner-selection ;
```

Refer to Item 2 of this command specification for details of *object-type* and *object-name-selection*.

where *owner-selection* is either:

- The authorization IDs of one or more owners separated by commas, or
- A combination of characters and ? and \* symbols (each symbol representing one or a string of characters) which together match the authorization IDs of a selection of owners and identifies the authorization IDs of the owners of the objects.

If you do not specify a CREATOR clause then the owner's authorization ID is taken to be the same as your ASG-ControlManager Logon Identifier. You cannot import information about an object if this differs from the authorization ID of the owner of the object.

### **Representing a Selection of Object Names and Owner Authorization IDs with ? and \* Symbols**

With a single EXTRACT command you can import information about all or a selection of objects owned by any or all owners.

To do so, use the ? and \* symbols to match the names of objects and the authorization IDs of the owners of those objects.

The ? symbol represents any single character. For example, if you specify an object name and an authorization ID of ???? in an EXTRACT command then all objects with a four character name owned by any owner with a four character authorization ID are selected.

The \* symbol represents any string of zero or more characters. For example, if you specify an object name and an authorization ID of \* in an EXTRACT command then all objects owned by any owner are selected.

You can combine ? and \* symbols together and with other characters. For example an object name of ?T\* represents all object names of any character length that have T as their second character.

You cannot represent object names or owner authorization IDs with ? and \* symbols in the same clause of an EXTRACT command in which you have also specified a list of object names or authorization IDs separated by commas.

### **Parent, Children, and Referenced Objects**

An object specified in an EXTRACT command is known as a *parent object*. A parent object can refer to other objects which are known as its *children*.

The information imported about a parent object can be used to generate a dictionary member. The member's definition will include clauses defining the parent object's relationship to its children.

The EXTRACT command will also import information about those children that *do not* have children of their own. This information can then be used to document the children as members in the dictionary.

Ownership of the parent object can be documented in the dictionary and is therefore included among the parent object's children.

Children that can have children of their own are known as *referenced objects*. The EXTRACT command does not import information about referenced objects. Referenced objects are added to the dictionary as dummy members referenced by the definition of the parent object.

If members with the same name as the proposed members documenting the children and referenced objects already exist in the dictionary then a relationship is created in the dictionary between the existing members and the member documenting the parent object.

For example, if you import information about a table you can document the table, and its columns, and ownership in the dictionary. Tables with which it has referential constraints would be referenced objects added to the dictionary as dummy members.

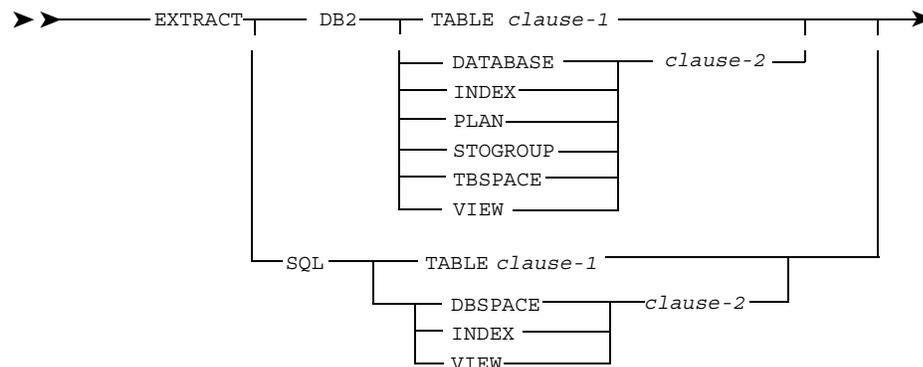
The EXTRACT command does not produce any output other than messages naming the parent objects on which information has been imported. Use the RECONCILE command to generate a Reconciliation Report listing the proposed members documenting the parent, children, and referenced objects.

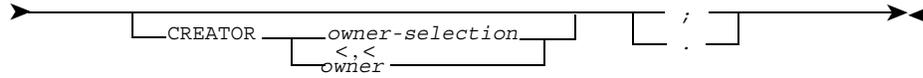
Information which can be Imported from SQL/DS

Table 17 Information Imported by the EXTRACT SQL Command

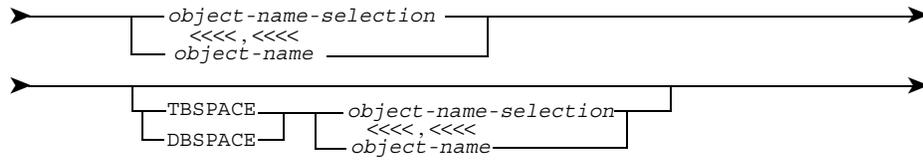
| EXTRACT Command     | Objects on which Information is Imported   | Dictionary Member Type   |
|---------------------|--|--|
| EXTRACT SQL TABLE   | A table and: <ul style="list-style-type: none"> <li>• Its owner</li> <li>• The dbspace it is in</li> <li>• Its columns</li> <li>• The tables with which it has referential constraints.</li> </ul>                           | SQL-TABLE<br>SQL-USER<br>dummy SQL-DBSPACE<br>ITEMs<br>dummy SQL-TABLES  |
| EXTRACT SQL VIEW    | A view and: <ul style="list-style-type: none"> <li>• Its owner</li> <li>• The columns that make up the view</li> <li>• The views upon which the view is based</li> <li>• The tables upon which the view is based.</li> </ul> | SQL-VIEW<br>SQL-USER<br>dummy ITEMs<br>dummy SQL-VIEW<br>dummy SQL-TABLE |
| EXTRACT SQL INDEX   | An index and: <ul style="list-style-type: none"> <li>• Its owner</li> <li>• The table it is indexing</li> <li>• The columns forming the index key.</li> </ul>  | SQL-INDEX<br>SQL-USER<br>dummy SQL-TABLE<br>dummy ITEMs                  |
| EXTRACT SQL DBSPACE | A dbspace and: <ul style="list-style-type: none"> <li>• Its owner.</li> </ul>  | SQL-DBSPACE<br>SQL-USER  |

Syntax of the EXTRACT Command





where *clause-1* is:



where:

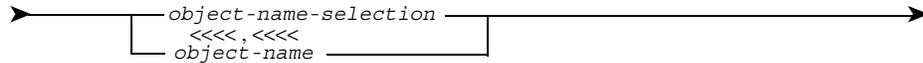
*object-name-selection* is a combination of characters and ? and \* symbols (each symbol representing a single or string of characters) which together match the names of a selection of external objects

*object-name* is the name of an external object.

where:

? represents a single character  
 \* represents a string of zero or more characters.

*clause-2* is:



where:

*object-name-selection* is as defined above  
*object-name* is as defined above.

where:

*owner-selection* is a combination of characters and ? and \* symbols (each symbol representing a single or string of characters) which together match the authorization IDs of a selection of owners

*owner* is the authorization ID of an owner.

where:

? is as defined above as defined above  
 \* is as defined above as defined above.

**Note:**

The DB2 and TBSPACE keywords are provided by the Corporate Dictionary/Repository Definition Import from DB2 facility.

The SQL and DBSPACE keywords are provided by the Corporate Dictionary/Repository Definition Import from SQL/DS facility.

---

## ISQL

Use the ISQL command to dynamically submit to your DB2 or SQL/DS environment, SQL statements entered in the Command printed in the current buffer, or filed in a USER-MEMBER.

Refer to "Syntax of the ISQL Command" on page 151 for the syntax of the ISQL command.

### Submitting SQL Statements

To dynamically submit to your relational environment 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 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.

You can also dynamically submit to your relational environment SQL statements printed in the current buffer or filed in a USER-MEMBER.

To dynamically submit to your relational environment SQL statements printed in the current buffer, enter:

```
ISQL ;
```

The current buffer can be a Command Mode, Edit, Update, or Lookaside Buffer.

To dynamically submit to your relational environment 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 and SQL CREATE commands. The CREATE TABLE statement can be either printed or, if you have specified an ONTO clause in the DB2 or SQL command, 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 by an ONTO clause do not include messages. You can also use the SWITCH MESSAGES command to stop Manager Products messages being displayed in the current buffer.

### **Restricting the Size 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 "Creating and Populating a Table" on page 115 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:

```
ISQL HELP sql/ds-sqlcode ;
```

or

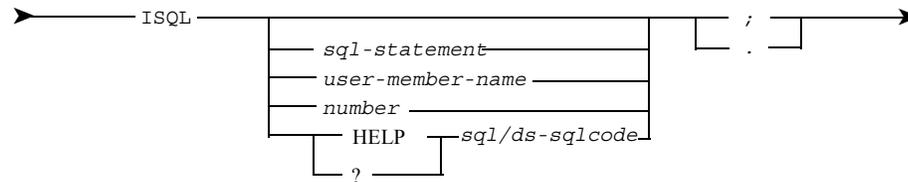
```
ISQL ? sql/ds-sqlcode ;
```

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 Printed by Dynamic SQL Services" on page 108 for details of the output printed by the ISQL command.

### Syntax of the ISQL Command



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

*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

Use the POPULATE command to enter in the dictionary the ADD or REPLACE commands and member definition statements generated from the information on the Workbench Translation Area documenting external objects.

Refer to "Syntax of the POPULATE Command" on page 152 for the syntax of the POPULATE command.

### Entering Statements into the Dictionary

Use the POPULATE command to enter into the dictionary the ADD or REPLACE command and member definition statements generated by a previous PREVIEW command. The statements are those displayed in the current buffer or filed in a USER-MEMBER.

To enter into the dictionary 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, 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 enter in the dictionary statements filed in a USER-MEMBER, enter:

```
POPULATE FROM 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 command have been filed.

You can also enter 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 NOPRINT command.

### **Specifying that Statements Will Form a Logical Unit of Work**

You can specify that the command and member definition statements entered in the dictionary 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 dictionary or be rolled back in its entirety, leaving the dictionary unchanged, if for any reason any of the statements are unsuccessful. In this way you can be sure that the statements documenting the parent object and all its children either have or have not been entered in the dictionary.

For example, you can avoid a situation where the definition of the parent object is entered in the dictionary but the definitions of some of its children are not.

Refer to *ASG-Manager Products Procedures Language* for details of Logical Units of Work.

### **Syntax of the POPULATE Command**

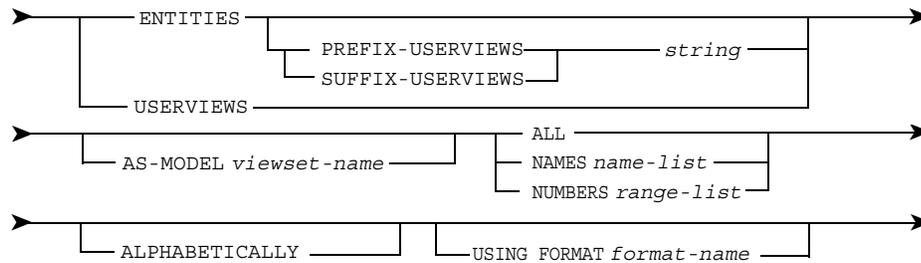
```
➤ POPULATE wbta-clause wbda-clause ;
```

where *wbta-clause* is:

```
➤ FROM BUFFER BUFFER user-member-name ROLLBACK
```

where *user-member-name* is the name of a USER-MEMBER.

*wbda-clause* is:



where:

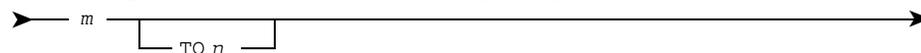
*string* can contain from 1 to 31 alphanumeric characters and must conform to the Manager Products rules for forming valid dictionary member names

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 thus producing the name of the supplementary USERVIEW. If the resulting userview name contains more than 32 characters, it is truncated appropriately.

*viewset-name* can contain from 1 to 32 alphanumeric characters and must conform to the Manager Products rules for forming valid dictionary names

*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 a list of numeric ranges, separated by commas,



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-name* is the name of a FORMAT member of the Modeling Dictionary. FORMAT members can be used only if the User Formatted Output facility is installed.

**Note:**

The *wbda-clause* is provided by ASG-DesignManager.

The *wbta-clause* is provided by the ASG-DictionaryManager Translation and Transfer Engine facility.

## PREVIEW

Use the PREVIEW command to generate ADD or REPLACE command statements and corresponding member definition statements from the information on the Workbench Translation Area documenting external objects.

Refer to "Syntax of the PREVIEW Command" on page 155 for the syntax of the PREVIEW command.

### **Generating Command and Member Definition Statements**

The PREVIEW 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 dictionary standards and invoke them in the USING clause of the PREVIEW command.

The member definition statements include a NOTE clause giving the time and date the statement was generated by the PREVIEW command and an ALIAS clause giving the name of the external object the definition is documenting.

If the definition is to replace an existing member then certain common clauses of the existing member are incorporated in the definition unless you have specified the NO-COMMON-CLAUSES keyword in a previous RECONCILE command.

PREVIEW 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 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 dictionary.

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 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 clause in the PREVIEW 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 dictionary using the POPULATE command.

Refer to "RECONCILE" on page 157 for further details of the common clauses generated in member definition statements.

### Generating Member Definition Statements in Your Own Layouts

You can tailor the PREVIEW command so that it generates member definition statements in layouts which suit your dictionary 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.

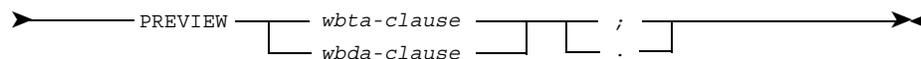
Alternatively you can tailor the Executive Routines in Manager Products' default layout rules.

Refer to "PREVIEW" on page 153 for further details of tailoring the PREVIEW command.

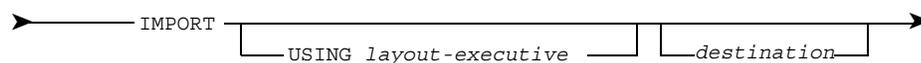
### Filing Generated Output in a USER-MEMBER

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO, PRIVATE, PUBLIC, NEW, APPEND, REPLACE, PRINT, and NOPRINT keywords.

### Syntax of the PREVIEW Command

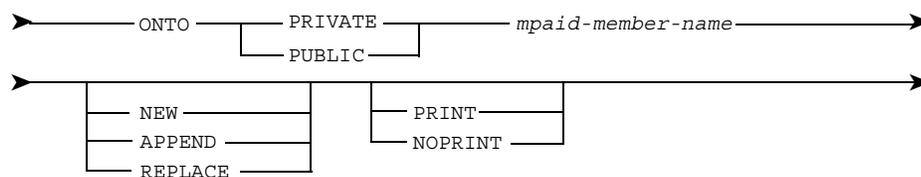


where *wbta-clause* is:

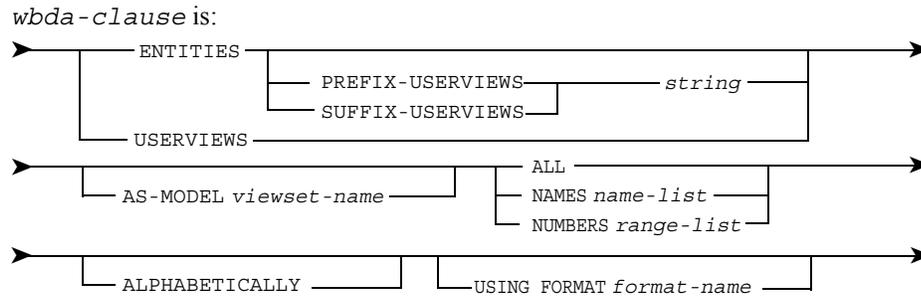


where *layout-executive* is the name of an Executive Routine.

*destination* is:



where *mpaid-member-name* is the name of a USER-MEMBER.



where:

*string* can contain from 1 to 31 alphanumeric characters and must conform to the Manager Products rules for forming valid dictionary member names

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 thus producing the name of the supplementary USERVIEW. If the resulting userview name contains more than 32 characters, it is truncated appropriately.

*viewset-name* can contain from 1 to 32 alphanumeric characters and must conform to the Manager Products rules for forming valid dictionary names

*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 a list of, numeric ranges, separated by commas, each of the form:



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-name* is the name of a FORMAT member of the Modeling Dictionary. FORMAT members can be used only if the User Formatted Output facility is installed.

**Note:** \_\_\_\_\_

The *wbda-clause* is provided by ASG-DesignManager.

The *wbta-clause* is provided by the ASG-DictionaryManager Translation and Transfer Engine facility.

## RADD

Use the RADD Line and Primary Commands to specify that you want a proposed member documenting an external object to be added to the dictionary.

**Use**

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 dictionary 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 dictionary by including an ADDING clause in the RECONCILE command.

Refer to "RECONCILE" on page 157 for further details of adding proposed members to the dictionary.

**Syntax of the RADD Command**

```
➤➤——RADD———▶▶
```

**Syntax of the RADD Primary Command**

```
➤➤——RADD n——□ ; □———▶▶
```

where *n* is a proposed member's identification number in a Reconciliation Report.

**RECONCILE**

Use the RECONCILE command to generate 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 dictionary.

Refer to "Syntax of the RECONCILE Command" on page 168 for the syntax of the RECONCILE command.

### **Introduction to 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 clause 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 clause 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 clause are generated for proposed members which have an ITEM member type.

The RECONCILE command does not update the dictionary but specifies the updates you intend to make. These updates are determined by the current contents of the dictionary. A proposed member is added to the dictionary unless a member (other than a dummy member without a source record) of the same name already exists.

Existing ITEM members are 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 is ignored.

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 dictionary. 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 dictionary members which have the same name.

You can use the Reconciliation Report to reconcile the proposed and existing dictionary members with one another. You can also use the RADD, RIGN, RREN, RREP, and RUPD commands during reconciliation.

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 dictionary. 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 dictionary 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 dictionary 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 dictionary 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 dictionary 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.

If the Basic or Advanced Status facilities are installed the updates to be made to the dictionary 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.

### ***Regenerating Proposed Members***

You can regenerate the proposed members documenting external objects.

To regenerate the proposed members, 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 dictionary member whose name is the same as a proposed member.

### ***Tailoring How Proposed Members are Generated***

You can tailor the RECONCILE command so that it generates proposed members which suit your dictionary standards.

To generate proposed members suiting your 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 dictionary 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 "RECONCILE" on page 157 for further details of tailoring the RECONCILE command.

### **Stopping Proposed Members being Entered in the Dictionary**

You can ignore a selection of proposed members so that they are not subsequently entered into the dictionary.

To ignore proposed members, enter:

```
RECONCILE IGNORING selection ;
```

where *selection* specifies which of the proposed members are to be ignored.

Refer to "Selecting Proposed Members to be Ignored, Added, or Replaced" on page 162 of this command specification 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*. Referenced objects are added to the dictionary as dummy members by a reference from the member documenting the parent object.

### **Adding Proposed Members**

You can specify that you want a selection of proposed members to be added to the dictionary.

To add proposed members, enter:

```
RECONCILE ADDING selection ;
```

where *selection* specifies which of the proposed members you want to be added to the dictionary.

Refer to "Selecting Proposed Members to be Ignored, Added, or Replaced" on page 162 of this command specification 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 dictionary 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 dictionary.

You cannot specify that proposed members documenting referenced objects are to be added to the dictionary. Referenced objects are added to the dictionary as dummy members by a reference from the member documenting the parent object.

### **Replacing Existing Members with Proposed Members**

You can specify that you want a selection of proposed members to replace existing dictionary members.

To replace existing members, enter:

```
RECONCILE REPLACING selection ;
```

where *selection* specifies which proposed members are to replace existing members.

Refer to "Selecting Proposed Members to be Ignored, Added, or Replaced" on page 162 of this command specification for details of *selection*.

The Reconciliation Report will indicate which of the proposed members are to replace existing dictionary 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 dictionary members. A relationship is created in the dictionary between the proposed member documenting the parent object and the existing member.

### **Renaming Proposed Members**

You can rename a selection of proposed members.

To rename proposed members, enter:

```
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 and

*member-name-2* is the new name

*n* is the proposed member's identification number in the Reconciliation Report.

You can rename as many proposed members as you want with a single RECONCILE command by repeating the MEMBER and NUMBER clauses. 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 clause 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
- 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 clause 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 Proposed Members to be Ignored, Added, or Replaced**

You can select which proposed members you want to be ignored, added to the dictionary, or replace existing dictionary members.

To select proposed members by their member type, enter:

```
RECONCILE update TYPE member-type-list ;
```

where:

*update* is IGNORING, REPLACING, or ADDING

*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 dictionary member, enter:

```
RECONCILE update GROUP DUPLICATES ;
```

The different updates and selections can be combined in a single RECONCILE command. For example, the following command:

```
RECONCILE REPLACING GROUP DUPLICATES IGNORING TYPE MODULE  
MEMBER IT-DEPT-NAME ;
```

specifies that:

- Proposed members will replace existing dictionary members which have the same member name

and

- 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 dictionary.

### **Excluding Common Clauses from the Definition of Proposed Members**

You can stop the common clauses of existing dictionary members being incorporated in the definitions of proposed members which are replacing them.

To exclude common clauses, enter:

```
RECONCILE NO-COMMON-CLAUSES ;
```

You can specify with a RECONCILE REPLACING or RREP command that a proposed member is to replace an existing dictionary member.

If you do not enter a RECONCILE command including the NO-COMMON-CLAUSES keyword then the ADMINISTRATIVE-DATA, ALIAS, COMMENT, DESCRIPTION, and NOTE common clauses of the existing dictionary member are incorporated in the definition of the proposed member replacing it.

Proposed members always have a NOTE and an ALIAS clause. The clauses are displayed in the member definition statements generated for the proposed members by a subsequent PREVIEW command.

The NOTE clause contains the date and time that the member definition statement was generated by the PREVIEW command. The ALIAS clause 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 clauses of the existing member is incorporated in those of the proposed member. The single ALIAS clause 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 common clauses 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 if you do not request a particular type of Reconciliation Report.

To display a detailed Reconciliation Report excluding certain information about the relationships between objects, enter:

```
RECONCILE LIST DETAILS NO-XREF ;
```

If you specify the X-REF keyword then:

- The table listing the children of the parent object and
- 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

are not displayed in the detailed Reconciliation Report.

### ***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 dictionary is to be updated with the proposed members. If ADD is specified, the proposed member is to be added to the dictionary. If REP is specified the proposed member is to replace an existing member in the dictionary. If IGN is specified the proposed member is not to be entered in the dictionary. A \* symbol indicates that the proposed member is a referenced object and is added to the dictionary as a dummy member by a reference from the member documenting the parent object.

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 indicates whether there is an existing dictionary 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. The entry \*NO AUTH is displayed if the Audit and Security facility is installed and 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.

*An Example of the Reconciliation Summary Report*

The following is an example of a Reconciliation Summary Report:

```
* * * * *
*       Reconciliation summary report       *
* for extract of table AAW.SALES from SQL/DS. *
* * * * *
```

| ID | Proposed Member name | Type        | Upd | Condition |
|----|----------------------|-------------|-----|-----------|
| 1  | TB-AAW-SALES         | SQL-TABLE   | ADD |           |
| 2  | US-AAW               | SQL-USER    | IGN | SCE-ENC   |
| 3  | DBS-NORTH            | SQL-DBSPACE | *   |           |
| 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   |

*An Example of the Reconciliation Detailed Report*

The following is an example of a Reconciliation Detailed Report:

```

* * * * *
*      Reconciliation summary report      *
* for extract of table AAW.SALES from SQL/DS. *
* * * * *

1  Extracted...SALES                TABLE
   Proposed...TB-AAW-SALES          SQL-TABLE      ADD
-----
-----6 CHILDREN extracted with AAW.SALES-----
-----
      1  CREATOR
      1  DBSPACE
      4  COLUMNS
-----

2  Extracted...  AAW                CREATOR
   Proposed...  US-AAW              SQL-USER      IGN
   Dictionary... SCE ENC            SQL-USER

3  Refers to...  NORTH              DBSPACE
   Proposed...  DBS-NORTH           SQL-DBSPACE  *

4  Extracted...  QTY                COLUMN
   Proposed...  IT-QTY              ITEM        ADD
   Dictionary... *  DUM              ITEM

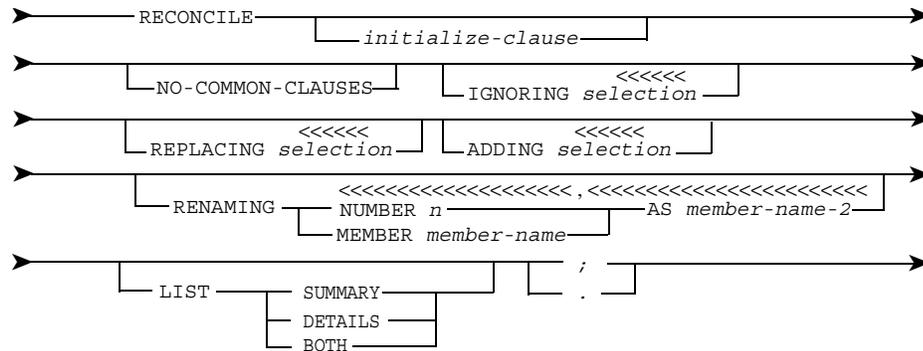
5  Extracted...  DESCRIPTION        COLUMN
   Proposed...  IT-DESCRIPTION       ITEM        IGN
   Form desc...  CHARACTERS 5        VERSION 1
   Dictionary... SCE ENC            ITEM
   Version 1...  HELD-AS CHARACTERS 5
   Version 2...  ENTERED-AS CHARACTERS 4

6  Extracted...  COST                COLUMN
   Proposed...  IT-DELIVERY          ITEM        ADD
   Form desc...  CHARACTERS 10 USAGE DATE VERSION 1

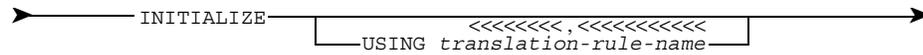
7  Extracted...  PRICE                COLUMN
   Proposed...  IT-PRICE              ITEM        REP
   Form desc...  NUMERIC 6           VERSION 3
   Dictionary... SCE ENC            ITEM
   Version 1...  HELD-AS CHARACTERS 5
   Version 2...  ENTERED-AS CHARACTERS 4
* * * * *

```

**Syntax of the RECONCILE Command**

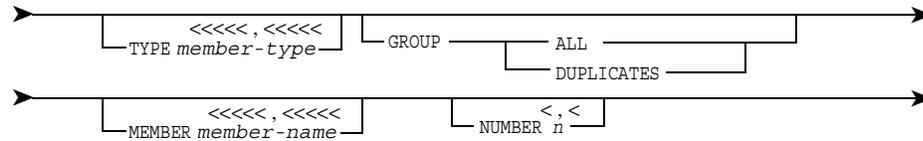


where *initialize-clause* is:



where *translation-rule-name* is the name of an Executive Routine.

*selection* is:



where:

*member-type* is the member type of a proposed member

*member-name* is the name of a proposed member

*n* is a proposed member's identification number in a Reconciliation Report.

**RIGN**

Use the RIGN Line or Primary Command to specify that you do not want a proposed member documenting an external object to be entered in the dictionary.

**Use**

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 clause in the RECONCILE command.

Refer to "RECONCILE" on page 157 for further details of ignoring proposed members.

**Syntax of the RIGN Line Command**



**Syntax of the RIGN Primary Command:**



where *n* is a proposed member’s identification number in a Reconciliation Report.

**RREN**

Use the RREN Line or Primary Command to rename a proposed member documenting an external object.

**Use**

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 clause in a RECONCILE command.

Refer to "RECONCILE" on page 157 for further details of renaming proposed members.

### Syntax of the RREN Line Command



### Syntax of the RREN Primary Command:



where *n* is a proposed member's identification number in a Reconciliation Report.

## RREP

Use the RREP Line or Primary Command to specify that you want a proposed member documenting an external object to replace an existing dictionary member.

### Use

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 dictionary member by including a REPLACING clause in the RECONCILE command.

Refer to "RECONCILE" on page 157 for further details of replacing existing members with proposed members.

### Syntax of the RREP Line Command

➤ — RREP — ➤

### Syntax of the RREP Primary Command:

➤ — RREP *n* [ ] ; [ ] — ➤

where *n* is a proposed member's identification number in a Reconciliation Report.

## RUPD

Use the RUPD Line or Primary Command to update an existing dictionary member from a Reconciliation Report in order to interactively change its source record.

### Use

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 dictionary 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 clause of the proposed member into the Update Buffer by using the I (Insert), F (Follow), and P (Precede) Line Commands.

To enter the contents of the buffer into the dictionary use the FILE or SFILE commands. To abandon the update without adding the contents to the dictionary 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 dictionary by the systems Administrator. Use the QUERY BUFFER-LIMIT command to find out the buffer limit.

The I, F, and P Line Commands only copy the form-description and USAGE clause 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.

If the Basic or Advanced Status facilities are installed 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, and XQUIT commands.

Refer to the *ASG-ControlManager User's Guide* for details of the QUERY BUFFER-LIMIT command.

### **Syntax of the RUPD Line Command**

➤ — RUPD ————— ➤

### **Syntax of the RUPD Primary Command:**

➤ — RUPD *n* [ ] ; [ ] ————— ➤

where *n* is a proposed member's identification number in a Reconciliation Report.

## **SQL ACQUIRE**

Use the SQL ACQUIRE command to generate an SQL ACQUIRE DBSPACE statement from the definition of an SQL-DBSPACE member.

### **Use**

To generate an SQL ACQUIRE DBSPACE statement, enter:

```
SQL ACQUIRE member-name ;
```

where *member-name* is an encoded SQL-DBSPACE member.

The generated SQL ACQUIRE statement can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file the SQL ACQUIRE statement in a USER-MEMBER you must specify an ONTO clause in the SQL ACQUIRE command.

The Systems Administrator can tailor the output of the SQL ACQUIRE command so that:

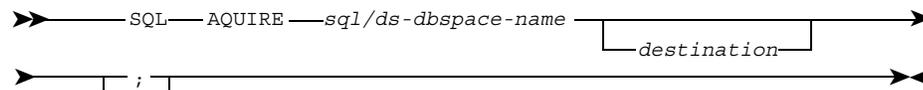
- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed.

Refer to "SQL-DBSPACE" on page 184 for an example of an SQL ACQUIRE DBSPACE statement generated from an SQL-DBSPACE member by the SQL ACQUIRE command.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

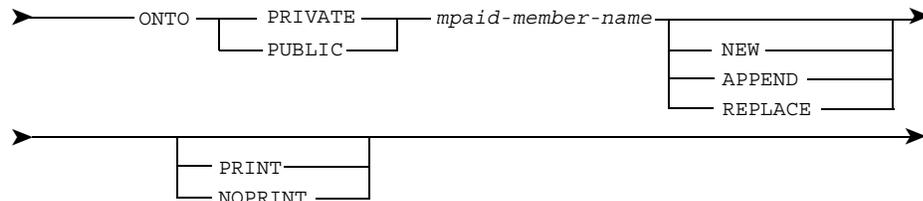
Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

### Syntax of the SQL ACQUIRE Command



where *sql/ds-dbpace-name* is the name of a SQL DBSPACE member.

*destination* is:



where *mpaid-member-name* is the name of a USER-MEMBER.

### SQL ALTER

Use the SQL ALTER command to generate one or more SQL ALTER TABLE statements from the definition of a SQL-TABLE member.

Refer to "Syntax of the SQL ALTER Command" on page 179 for the syntax of the SQL ALTER command.

### **Introduction to the SQL ALTER Command**

Use the SQL ALTER command to generate one or more SQL ALTER TABLE statements, from the definition of a SQL-TABLE member, which if applied to your SQL/DS environment will make the following alterations to a table:

- Add one or more columns
- Add or drop a primary key
- Add or drop a referential constraint
- Activate or deactivate a primary key
- Activate or deactivate a foreign key
- Activate or deactivate a primary key and all foreign keys
- A combination of the above.

The generated SQL ALTER TABLE statements can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file SQL ALTER TABLE statements in a USER-MEMBER you must specify an ONTO clause in the SQL ALTER command.

The different SQL ALTER TABLE statements are generated from particular clauses in the definition of the SQL-TABLE member specified in the SQL ALTER command. The SQL ALTER command does not alter the SQL-TABLE member's definition.

By accurately documenting a table in the definition of a SQL-TABLE member you can maintain the table with SQL statements generated from the definition.

When generating SQL ALTER TABLE statements to add a column, primary key, or referential constraint, or to activate or deactivate primary and foreign keys you should add the clauses from which the SQL statements will be generated, before entering the SQL ALTER command.

If the clauses are not present an SQL statement will not be generated. The generated SQL statement could also be rejected when applied to your SQL/DS environment if the clauses are present but do not accurately document the table.

When generating an SQL ALTER TABLE statement to drop a referential constraint or primary key you should drop the clauses from which the SQL statement will be generated, after entering the SQL ALTER command. If the clauses have already been removed the SQL statement will not be generated.

The Systems Administrator can tailor the output of the SQL ALTER command so that:

- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed.

Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

### **Generating SQL Statements to Add Columns to a Table**

Use the SQL ALTER command to generate one or more SQL ALTER TABLE statements which if applied to your SQL/DS environment will add columns to a table.

To generate SQL ALTER TABLE statements to add columns to a table, enter:

```
SQL ALTER member-name ADD COLUMNS n ;
```

where:

*member-name* is an encoded SQL-TABLE member

*n* is a number from 1 to 299 identifying 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 clause of the SQL-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 clause.

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 SQL/DS 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 clause 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 SQL/DS data type of columns generated in SQL ALTER TABLE statements is derived from the definition of the ITEMS and GROUPS specified in the COLUMNS clause.

SQL/DS requires that new columns added to a table must allow a null or default value. The SQL ALTER command therefore displays a warning message if any of the columns are defined in the SQL-TABLE member definition as being NOT-NULL.

Refer to "Generating Column Data Types" on page 96 for details of generating column data types.

Refer to "Member Type and Command Descriptions" on page 143 for details of the COLUMNS clause.

Refer to "Member Type and Command Descriptions" on page 143 for details of the NOT-NULL keyword.

### **Generating an SQL Statement to Add or Drop a Primary Key on a Table**

Use the SQL ALTER command to generate an SQL ALTER TABLE statement which if applied to your SQL/DS environment will add or drop a primary key on a table.

To generate an SQL ALTER TABLE statement to add a primary key, enter:

```
SQL ALTER member-name ADD PRIMARY-KEY ;
```

where *member-name* is an encoded SQL-TABLE member.

To generate an SQL ALTER TABLE statement to drop a primary key, enter:

```
SQL ALTER member-name DROP PRIMARY-KEY ;
```

Primary key columns are defined with the PRIMARY-KEY keyword in the SQL-TABLE member. All columns in the SQL-TABLE that have an associated PRIMARY-KEY keyword are generated as part of the primary key.

Columns which allow a null value cannot be part of the primary key. The SQL ALTER command therefore displays a warning message if any of the columns defined in the SQL-TABLE member as being part of the primary key do not have an associated NOT-NULL keyword.

The SQL statement will be rejected when submitted to SQL/DS 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 SQL ALTER command to drop the existing primary key and then to add the new one.

If you have replaced the primary key on a parent table you may also need to change the foreign keys of dependent tables.

### **Generating an SQL Statement to Add or Drop a Referential Constraint on a Table**

Use the SQL ALTER command to generate an SQL ALTER TABLE statement which if applied to your SQL/DS environment will add or drop a referential constraint on a table.

To generate an SQL ALTER TABLE statement to add a referential constraint, enter:

```
SQL ALTER member-name ADD CONSTRAINT NAMED constraint-name ;
```

or

```
SQL ALTER member-name ADD CONSTRAINT NUMBER n ;
```

where:

*member-name* is an encoded SQL-TABLE member

*constraint-name* is a name specified in the NAMED clause of the SQL-TABLE member which identifies the referential constraint

*n* is a number identifying the referential constraint by its sequence among other referential constraints in the SQL-TABLE member definition.

To generate an SQL ALTER TABLE statement to drop a referential constraint, enter:

```
SQL ALTER member-name DROP CONSTRAINT NAMED constraint-name ;
```

or

```
SQL ALTER member-name DROP CONSTRAINT NUMBER n ;
```

Referential constraints are defined in the CONSTRAINT clauses of SQL-TABLE members. The SQL ALTER TABLE statement can be generated from a particular CONSTRAINT clause. A SQL-TABLE member can have any number of CONSTRAINT clauses; to identify which clause you want to generate you must select it by its name or sequence within the member's definition.

For example, the command:

```
SQL ALTER CUST-TABLE ADD CONSTRAINT NUMBER 3 ;
```

would generate an SQL ALTER TABLE statement to add a referential constraint on a table from the third CONSTRAINTS clause in the definition of the member CUST-TABLE.

The SQL statement will be rejected when submitted to SQL/DS 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 first use the SQL ALTER command to drop the existing referential constraint and then to add the new one.

Refer to "Member Type and Command Descriptions" on page 143 for details of the CONSTRAINT and NAMED clauses.

### **Generating an SQL Statement to Activate or Deactivate a Primary Key on a Table**

Use the SQL ALTER command to generate an SQL ALTER TABLE statement which if applied to your SQL/DS environment will activate or deactivate a primary key on a table.

To generate an SQL ALTER TABLE statement to activate a primary key, enter:

```
SQL ALTER member-name ACTIVATE-PRIMARY-KEY ;
```

where *member-name* is an encoded SQL-TABLE member.

To generate an SQL ALTER TABLE statement to deactivate a primary key, enter:

```
SQL ALTER member-name DEACTIVATE-PRIMARY-KEY ;
```

Primary key columns are defined with the PRIMARY-KEY keyword in the SQL-TABLE member.

Activating and deactivating a primary key on a parent table will also implicitly activate or deactivate foreign keys on its dependent tables.

You can generate an SQL ALTER TABLE statement to explicitly activate or deactivate a particular foreign key.

You can also generate an SQL ALTER TABLE statement to activate or deactivate the primary and foreign keys on a table and all the foreign keys on its dependent tables.

### Generating an SQL Statement to Activate or Deactivate a Foreign Key on a Table

Use the SQL ALTER command to generate an SQL ALTER TABLE statement which if applied to your SQL/DS environment will activate or deactivate a foreign key on a table.

To generate an SQL ALTER TABLE statement to activate a foreign key, enter:

```
SQL ALTER member-name ACTIVATE CONSTRAINT NAMED constraint-name ;
```

or

```
SQL ALTER member-name ACTIVATE CONSTRAINT NUMBER n ;
```

where:

*member-name* is an encoded SQL-TABLE member

*constraint-name* is a name specified in the NAMED clause of the SQL-TABLE member which identifies the referential constraint defining the foreign key

*n* is a number identifying the referential constraint by its sequence among the other referential constraints in the SQL-TABLE member definition.

To generate an SQL ALTER TABLE statement to deactivate a foreign key, enter:

```
SQL ALTER member-name DEACTIVATE CONSTRAINT NAMED constraint-name ;
```

or

```
SQL ALTER member-name DEACTIVATE CONSTRAINT NUMBER n ;
```

Foreign keys are defined in the CONSTRAINT clauses of SQL-TABLE members. The SQL ALTER TABLE statement can be generated from a particular CONSTRAINT clause. A SQL-TABLE member can have any number of CONSTRAINT clauses: to identify which clause you want to generate you must select it by its name or sequence within the member's definition.

You can also generate an SQL ALTER TABLE statement to activate or deactivate the primary and foreign keys on a table and all the foreign keys on its dependent tables.

### Generating an SQL Statement to Activate or Deactivate Both a Primary Key and All Foreign Keys

Use the SQL ALTER command to generate an SQL ALTER TABLE statement which if applied to your SQL/DS environment will activate or deactivate both the primary and foreign keys on a parent table and all the foreign keys on its dependent tables.

To generate an SQL ALTER TABLE statement to activate both the primary key and all foreign keys, enter:

```
SQL ALTER member-name ACTIVATE ALL ;
```

where *member-name* is an encoded SQL-TABLE member.





where *member-name* is an encoded SQL-TABLE or SQL-VIEW member.

The generated SQL COMMENT ON and LABEL ON statements can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file SQL COMMENT ON or LABEL ON statements in a USER-MEMBER you must specify an ONTO clause in the SQL COMMENT or SQL LABEL command.

SQL COMMENT ON and SQL LABEL ON statements can be generated for a table or view, and for columns within a table or view. An SQL COMMENT ON statement is generated from every SQL-COMMENT clause and an SQL LABEL ON statement from every SQL-LABEL clause in the definition of the specified SQL-TABLE or SQL-VIEW member. If the member does not have a SQL-COMMENT or SQL-LABEL clause then no statements are generated.

The Systems Administrator can tailor the output of the SQL COMMENT and SQL LABEL commands so that:

- SQL/DS object names are derived from aliases
- Internal diagnostic is displayed.

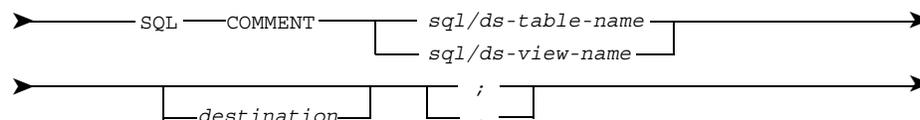
The Systems Administrator can also tailor the output of the SQL CREATE command so that SQL CREATE, COMMENT ON, and LABEL ON statements are generated from the same member at the same time.

Refer to "SQL-TABLE" on page 245 for an example of SQL CREATE, COMMENT ON, and LABEL ON statements generated from a SQL-TABLE member by the SQL CREATE command.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

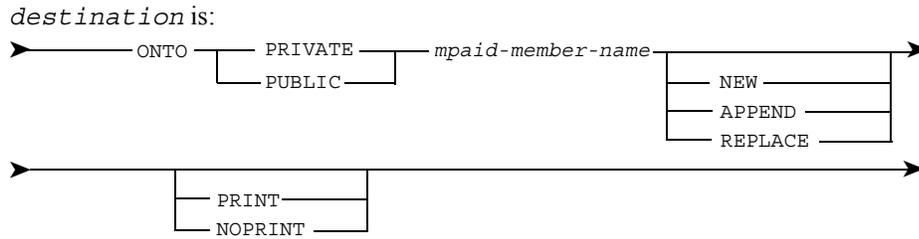
### Syntax of the SQL COMMENT Command



where:

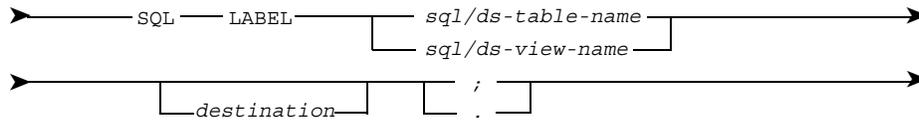
*sql/ds-table-name* is the name of a SQL-TABLE member

*sql/ds-view-name* is the name of a SQL-VIEW member.



where *mpaid-member-name* is the name of a USER-MEMBER.

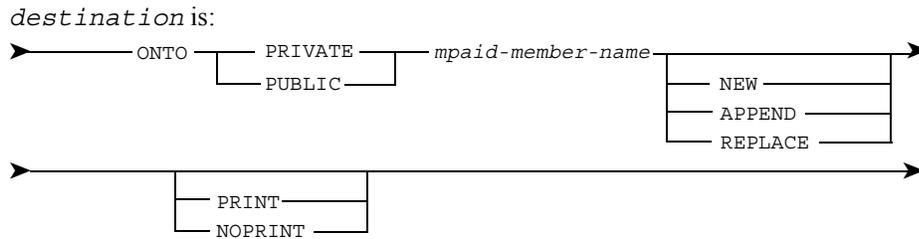
### Syntax of the SQL LABEL Command



where:

*sql/ds-table-name* is the name of a SQL-TABLE member

*sql/ds-view-name* is the name of a SQL-VIEW member.



where *mpaid-member-name* is the name of a USER-MEMBER.

## SQL CREATE

Use the SQL CREATE command to generate an SQL CREATE statement for a SQL/DS object from its definition in a dictionary member.

### Use

To generate an SQL CREATE statement, enter:

```
SQL CREATE member-name ;
```

where *member-name* is an encoded SQL-INDEX, SQL-TABLE, or SQL-VIEW member.

The SQL CREATE command generates an SQL CREATE INDEX, CREATE TABLE, or CREATE VIEW statement.

To generate SQL CREATE SYNONYM statements use the SQL SYNONYM command.

The generated SQL CREATE statement can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file the SQL CREATE statement in a USER-MEMBER you must specify an ONTO clause in the SQL CREATE command.

The SQL/DS data type of a column in a table is derived from the definition of the ITEMS and GROUPS specified in the COLUNMS clause of the SQL-TABLE member from which the SQL CREATE TABLE statement is being generated.

The Systems Administrator can tailor the output of the SQL CREATE command so that:

- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed
- SQL CREATE, COMMENT ON, and LABEL ON statements are generated from the same member at the same time.

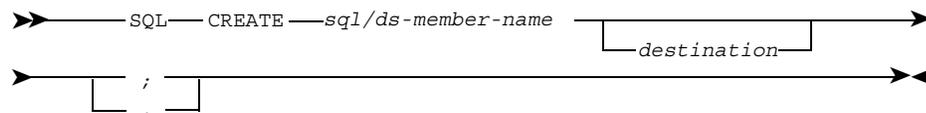
Refer to "SQL-TABLE" on page 245 for an example of an SQL CREATE statement generated from a SQL-TABLE member by the SQL CREATE command.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

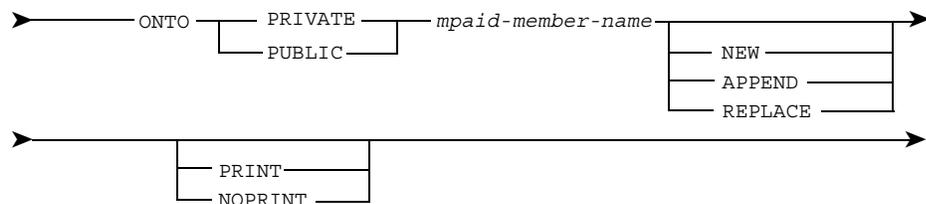
Refer to "Generating Column Data Types" on page 96 for details of generating column data types.

### Syntax of the SQL CREATE Command



where *sql/ds-member-name* is the name of a SQL-INDEX, SQL-TABLE, or SQL-VIEW member.

*destination* is:



where *mpaid-member-name* is the name of a USER-MEMBER.

## SQL-DBSPACE

SQL/DS dbspaces are defined in the dictionary as SQL-DBSPACE members.

Refer to "Syntax of the SQL-DBSPACE Member Type" on page 187 for the syntax of the SQL-DBSPACE member type.

### Introduction to SQL-DBSPACE

To document an SQL/DS dspace in the dictionary use the SQL-DBSPACE member type. To define the dictionary member type enter SQL-DBSPACE at the start of your member definition statement. Clauses are available to define the owner (the CREATOR-OWNER clause) and the type of the dspace. You can define the NHEADER, PAGES, PCTINDEX, PCTFREE, LOCK, and STORPOOL parameters in clauses with those names.

Clauses may be declared in any order.

**Note:** \_\_\_\_\_

The name you give to an SQL-DBSPACE member may be used as the SQL/DS object name.

---

### The AS Clause

Refer to "Defining an AS Clause" on page 277 for details of the AS clause.

### Defining the Owner of a Dspace

Use a CREATOR-OWNER clause to specify the owner of a dspace.

The syntax of the clause is as follows:

► \_\_\_\_\_ CREATOR-OWNER *sql-user*

where *sql-user* is the name of a dictionary member of the type SQL-USER, which represents the Authorization ID of the owner of the dspace. The owner of a dspace is usually the creator, but in SQL/DS Version 2 Release 2, the owner may not be the creator.

The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL ACQUIRE and DROP statements.

### Defining the Type of a Dspace

Use one of the keywords PUBLIC or PRIVATE to define whether a dspace is to be a public one or a private one, respectively. The keywords are the same as those used by SQL and they have the same meanings.

The clause is checked on encoding to ensure that only one of these options is specified.

If you omit to specify the type of a dbspace, the SQL default of PRIVATE will apply.

### Reserving Space for Header Pages

Use the NHEADER clause to specify the space to be reserved for header pages. The syntax of the clause is as follows:

➤ NHEADER *integer*

where *integer* is an integer in the range 1-8 inclusive, being the number of logical pages to be reserved. The clause is checked on encoding to ensure that the number of pages you specify is within the permitted range.

### Defining the Number of Pages Required for a Dbspace

Use a PAGES clause to define the minimum number of logical pages you want a dbspace to have. The syntax of the clause is as follows:

➤ PAGES *integer*

where *integer* is the minimum number of logical pages required.

### Defining Free Space for Indexes

Use a PCTINDEX clause to define the percentage of free space which you want to be left in a dbspace for indexes. The syntax of the clause is as follows:

➤ PCTINDEX *integer*

where *integer* is an integer of between 0 and 99. The clause is checked on encoding to ensure that the percentage you specify is within the permitted range.

### Defining Free Space

Use a PCTFREE clause to define the percentage of free space which you want to be left on each page of the dbspace. The syntax of the clause is as follows:

➤ PCTFREE *integer*

where *integer* is an integer in the range 0 to 99. The clause is checked on encoding to ensure that the percentage you specify is within the permitted range.

### Defining the Locking Level

Use a LOCK clause to define the locking level for a dbspace.

The syntax of the clause is as follows:

➤ LOCK 

|         |
|---------|
| PAGE    |
| DBSPACE |
| ROW     |

where PAGE, DBSPACE, and ROW are the permitted locking levels. The clause is checked on encoding to ensure that the option specified is a valid one.

**Note:**

You may use the LOCK clause only if you have defined the dbspace as a PUBLIC one.

**Defining the Storage Pool where Space is to be Acquired**

Use a STORPOOL clause to define the number of the storage pool where a dbspace is to be acquired. The syntax of the clause is as follows:

STORPOOL *integer*

where *integer* is the number of the storage pool.

**Example: SQL-DBSPACE Definition and SQL Generation**

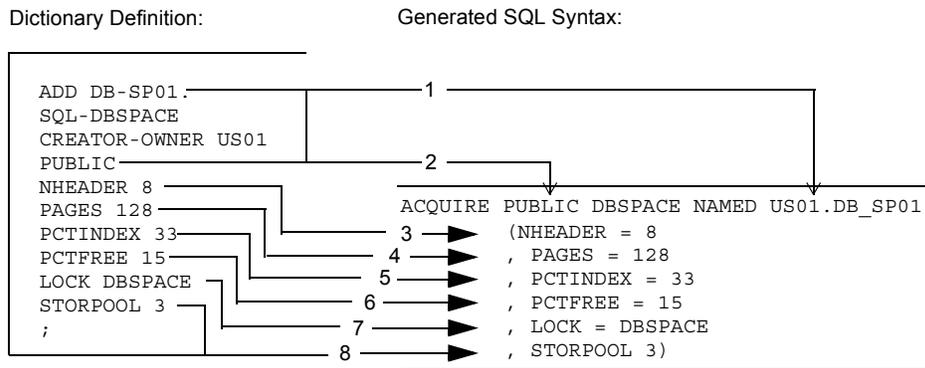
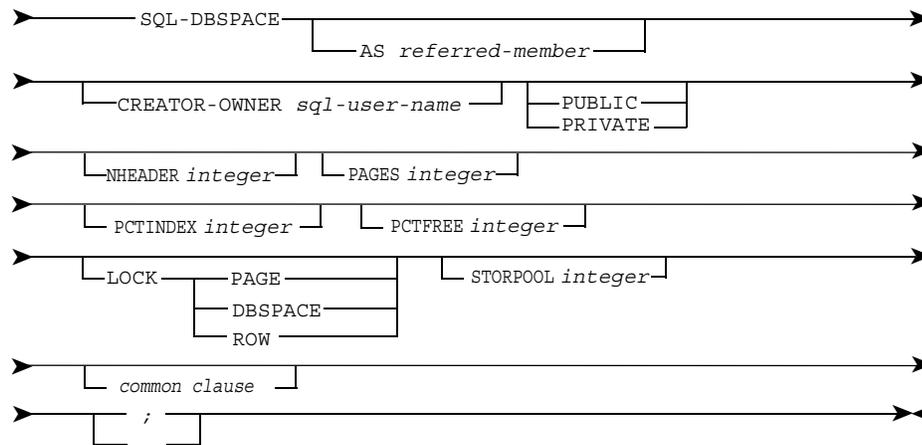


Figure 33 SQL-DBSPACE Definition and SQL Generation.

1. The dbspace name is taken from the dictionary definition for the dbspace, according to the rules for the derivation of external names. It is qualified by the name of the CREATOR-OWNER.
2. The dbspace type PUBLIC is taken directly from the dictionary definition.
- 3, 4, 5, 6, 7, 8 The NHEADER, PAGES, PCTINDEX, PCTFREE, LOCK, and STORPOOL parameters are taken directly from the dictionary definition.

**Syntax of the SQL-DBSPACE Member Type**

where:

*referred-member* is the name of an SQL-DBSPACE dictionary member

*sql-user-name* is the name of an SQL-USER dictionary member

*integer* (in the NHEADER clause) is an integer in the range 1 to 8

*integer* (in the PAGES clause) is an integer within the permitted range, being the number of pages required for the dbspace

*integer* (in the PCTINDEX clause) is an integer in the range 0 to 99

*integer* (in the PCTFREE clause) is an integer in the range 0 to 99

*integer* (in the STORPOOL clause) is an integer within the permitted range, being the number of the storage pool in which the dbspace is to be acquired.

**SQL DROP**

Use the SQL DROP command to generate an SQL DROP statement. An impact analysis report is generated displaying the impact the SQL DROP statement will have in your SQL/DS environment.

Refer to "Syntax of the SQL DROP Command" on page 190 for the syntax of the SQL DROP command.

**Generating an SQL DROP Statement**

To generate an SQL DROP statement, enter:

```
SQL DROP member-name ;
```

where *member-name* is an encoded SQL-DBSPACE, SQL-INDEX, SQL-TABLE, or SQL-VIEW member.

The SQL DROP command generates an SQL DROP DBSPACE, DROP INDEX, DROP TABLE, or DROP VIEW statement.

To generate SQL DROP SYNONYM statements use the SQL SYNONYM command.

The generated SQL DROP statement can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file the SQL DROP statement in a USER-MEMBER you must specify an ONTO clause in the SQL DROP command.

An SQL DROP statement will, when applied to your SQL/DS environment, drop both the specified SQL/DS object and other SQL/DS objects dependent upon it. For example, if you drop a table then all views and indexes dependent on that table are also dropped.

The SQL DROP command therefore also generates an impact analysis report displaying the impact the SQL DROP statement will have in your SQL/DS environment.

The report is always printed unless the keyword NO-IMPACT has been specified in the command. You cannot automatically file impact analysis reports in a USER-MEMBER.

The Systems Administrator can tailor the output of the SQL DROP command so that:

- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed.

Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

### ***The Impact Analysis Report***

The SQL DROP command enables you to assess the impact of the SQL DROP statement you have generated by also generating an impact analysis report displaying the impacted members in the dictionary, and therefore the SQL/DS objects they document, that would be dropped or affected were that SQL DROP statement to be applied to your SQL/DS environment.

An impact analysis report is always generated except when a SQL DROP command is applied to a SQL-INDEX member (as only the specified index would be dropped or affected by the SQL DROP INDEX statement), or the keyword NO-IMPACT has been specified in the command.

Members impacted by the SQL DROP command are reported but are not removed from the dictionary. When you have dropped the object from the SQL/DS environment you should (unless you intend at some future date to use the SQL CREATE command to recreate the object and those dependent on it) update the dictionary to reflect the changes. The impact analysis report will help you carry out these updates.

### How to Prevent an Impact Analysis Report being Generated

You can prevent an impact analysis report being generated by specifying the keyword NO-IMPACT in the SQL DROP command.

To generate an SQL DROP statement without an impact analysis report, enter:

```
SQL DROP member-name NO-IMPACT ;
```

where *member-name* is an encoded SQL-DBSPACE, SQL-INDEX, SQL-TABLE, or SQL-VIEW member.

An impact analysis report is not generated when a SQL DROP command is applied to a SQL-INDEX member.

### The Structure of the Impact Analysis Report

The impact analysis report has the following structure:

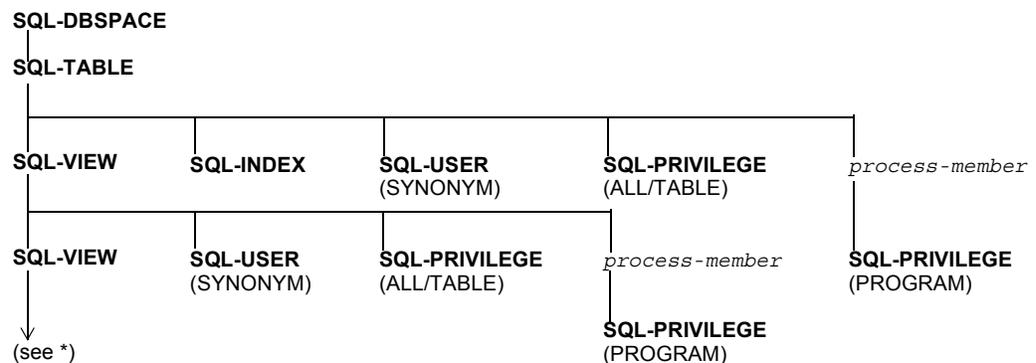


Figure 34 The Structure of the Impact Analysis Report

where *process-member* is a PROGRAM, MODULE, SYSTEM, or MMR-SYSTEM member.

For example, if a SQL-TABLE member is specified in a SQL DROP command then an SQL DROP TABLE statement is generated which if applied to your SQL/DS environment will drop that table and any views, indexes, and privileges dependent on it. Synonyms and SQL statements which refer to the dropped tables and views will be affected. SQL statements can be imbedded in programs which can in turn have privileges granted on them.

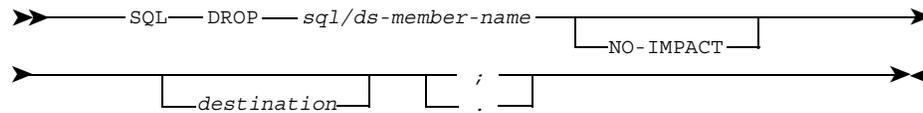
The impact analysis report will detail the SQL-TABLE, SQL-VIEW, SQL-INDEX, SQL-USER, SQL-PRIVILEGE, and process members that document the SQL/DS objects which will be dropped and the synonyms, programs, and program privileges that will be affected.

The impact analysis report therefore shows you the predicted effects in your SQL/DS environment of the SQL DROP statement you have generated from the dictionary.

### Filing Generated Output in a USER-MEMBER

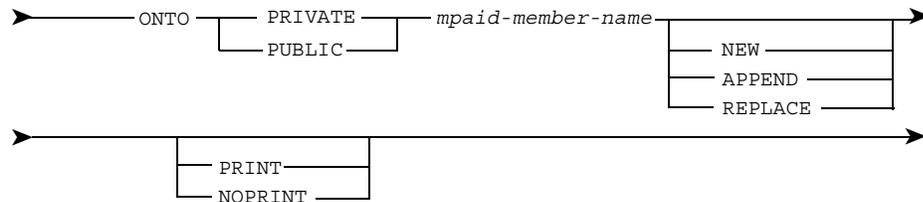
Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO, PRIVATE, PUBLIC, NEW, APPEND, REPLACE, PRINT, and NOPRINT keywords.

### Syntax of the SQL DROP Command



where *sql/ds-member-name* is the name of a SQL-DBSPACE, SQL-INDEX, SQL-TABLE, or SQL-VIEW member.

*destination* is:



where *mpaid-member-name* is the name of a USER-MEMBER.

### SQL GRANT and SQL REVOKE

Use the SQL GRANT and SQL REVOKE commands to generate respectively SQL GRANT and REVOKE statements from the definition of a SQL-PRIVILEGE member.

#### Use

To generate SQL GRANT statements, enter:

```
SQL GRANT member-name ;
```

To generate SQL REVOKE statements, enter:

```
SQL REVOKE member-name ;
```

where *member-name* is an encoded SQL-PRIVILEGE member.

The generated SQL GRANT and REVOKE statements can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file SQL GRANT or REVOKE statements in a USER-MEMBER you must specify an ONTO clause in the SQL GRANT or SQL REVOKE command.

A single SQL GRANT or SQL REVOKE command will generate a separate SQL GRANT or REVOKE statement for each SQL-TABLE, SQL-VIEW, or PROGRAM member specified in the ON clause of the SQL-PRIVILEGE member.

The generated SQL GRANT statements will include the WITH GRANT OPTION clause if the WITH-GRANT-OPTION keyword is present in the definition of the SQL-PRIVILEGE member.

SQL REVOKE statements cannot be generated with the BY clause as this would require keeping track of grantors who pass on a privilege they themselves have been granted with an SQL GRANT statement including the WITH GRANT OPTION.

The Systems Administrator can tailor the output of the SQL GRANT and SQL REVOKE commands so that:

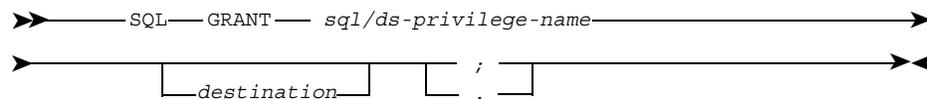
- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed.

Refer to "SQL-PRIVILEGE" on page 229 for an example of an SQL GRANT statement generated from an SQL-PRIVILEGE member.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

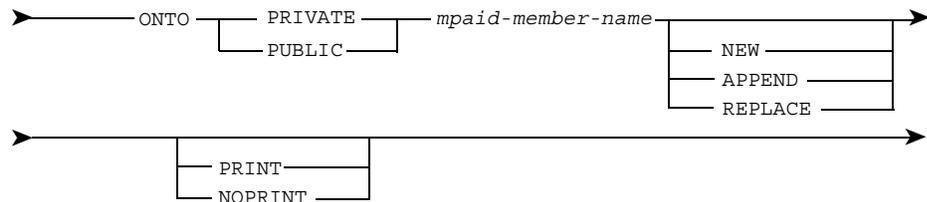
Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

**Syntax of the SQL GRANT Command**



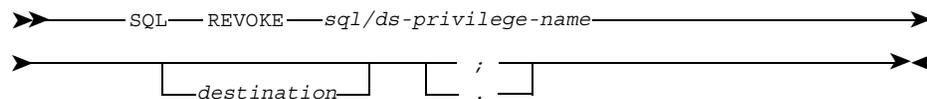
where *sql/ds-privilege-name* is the name of a SQL-PRIVILEGE member.

*destination* is:

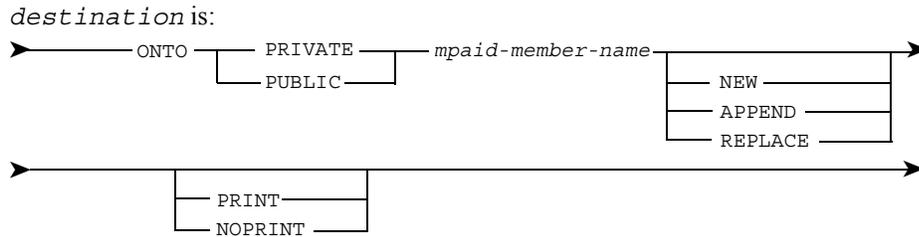


where *mpaid-member-name* is the name of a USER-MEMBER.

**Syntax of the SQL REVOKE Command**



where *sql/ds-privilege-name* is the name of a SQL-PRIVILEGE member.



where *mpaid-member-name* is the name of a USER-MEMBER.

## SQL-INDEX

SQL/DS indexes are defined in the dictionary as SQL-INDEX members.

Refer to "Syntax of the SQL-INDEX Member Type" on page 196 for the syntax of the SQL-INDEX member type.

### Introduction to SQL-INDEX

To document an SQL/DS index in the dictionary use the SQL-INDEX member type. To define the dictionary member type enter SQL-INDEX at the start of your member definition statement.

Clauses and keywords are available which allow you to specify the owner of an index (the CREATOR-OWNER clause), and the table to be indexed (the ON clause). You can specify the columns which are to be indexed (in a CONTAINS clause). The keyword UNIQUE is available to define an index as unique. You can define the percentage of free space to be left in an index (using the PCTFREE clause).

Clauses may be declared in any order.

The CREATOR-OWNER, ON, and CONTAINS clauses must be present for the successful generation of an SQL CREATE statement.

The CREATOR-OWNER clause is the only clause which must be present for the successful generation of an SQL DROP statement.

**Note:** \_\_\_\_\_

\_\_\_\_\_

The name you give to an SQL-INDEX member may be used as the SQL/DS object name.

\_\_\_\_\_

Refer to "Naming Conventions for SQL/DS Members" on page 89 for details of naming conventions, and the derivation of external names.

### The AS Clause

Refer to "Defining an AS Clause" on page 277 for details of the AS clause.

### Defining the Owner of an SQL-INDEX

Use a CREATOR-OWNER clause to specify the owner of an index. The syntax of the clause is as follows:

➤ CREATOR-OWNER *sql-user* ➤

where *sql-user* is the name of a dictionary member of the type SQL-USER, which represents the Authorization ID of the owner of the index. The owner of an index is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL requirements.

### Defining an Index as Unique

Use the keyword UNIQUE to define an index as unique, that is, the column or columns which are being indexed must not have duplicate entries in the table being indexed. If a single column is being indexed, then no value may appear more than once in that column. If more than one column is being indexed, then any given set of values can only appear once.

### Defining the Table to be Indexed

Use an ON clause to specify the table which is to be indexed. The format of the clause is as follows:

➤ ON *sql-table-name* ➤

where *sql-table-name* is the name of a dictionary member of the type SQL-TABLE. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL CREATE statements.

### Defining the Columns being Indexed

Use a CONTAINS clause to specify the column or columns which form the index key. The keyword CONTAINS is followed by one or more column-specifications. The syntax of the clause is as follows:

➤ CONTAINS *contains-clause* ➤

*contains-clause* is:

➤ *single-column-clause* ➤

┌ ( *integer* ) ─── *group-name* EXPAND ───┐

┌ ASCENDING ───┐

└ DESCENDING ───┘

*single-column-clause* is:

➤ PRIVATE ───┐

└ PUBLIC ───┘ ┌ KNOWN-AS *local-name* ───┐

where:

*(integer)* is the number of columns in a 'column set'. The column specified in the following single-column-clause will be repeated by the number of times you have specified. On generation of an SQL statement each column produced will be suffixed automatically by a number; the first column will be suffixed by 1, the second by 2, and so on. For example, suppose you have a set of two columns, PERIOD\_1 and PERIOD\_2. If you specify (2) PERIOD in the CONTAINS clause (where PERIOD is the name of the ITEM dictionary member which represents the column) two columns, PERIOD\_1 and PERIOD\_2, will be generated.

*item-name* is the name of a dictionary member of the type ITEM, which represents one column in the table being indexed

*group-name* in the single-column-clause is the name of a dictionary member of the type GROUP, which represents one column in the table being indexed.

Use a KNOWN-AS clause to specify a local-name which will be used as the name of the SQL/DS indexing column.

Use the group-name EXPAND option to refer to a collection of GROUPs and/or ITEMs, each of which represents a column being indexed. Group-name is the name of a dictionary member of the type GROUP.

**Note:** \_\_\_\_\_

Local-names may not be specified for EXPANDED members: this is checked on encoding.

---

You may specify a maximum of 16 columns in an index key. The clause is checked on generation to ensure that the number of columns specified does not exceed this maximum.

The CONTAINS clause is checked on encoding to ensure that the members specified are of the correct type (that is, GROUPs or ITEMs). The clause is checked on generation to ensure that the length of the derived names of the columns is compatible with the external environment.

The clause is checked on generation also to ensure that no duplicate column-names have been specified.

Each column-name in an index must have the same, corresponding, column-name in the table it indexes.

Use one of the keywords ASCENDING or DESCENDING to determine the sort-sequence for columns in the index. If you omit to specify a sort-sequence, none will be generated and the SQL default will apply.

### Defining Several Columns at Once in an SQL-INDEX

Use a group-name EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

———— *group-name* EXPAND ————>

where *group-name* is the name of a dictionary member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the index at once; it is a shorthand way of referring to a number of GROUPs and ITEMs (that is, all those contained by the specified GROUP), which will each represent one column in the index. Your installation may already have GROUPs defined in its dictionary which are used in existing applications. These GROUPs may represent records or segments that now need to have counterparts in the SQL/DS environment.

Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

A GROUP to be EXPANDED should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in an index must have a name unique in the index and cannot 'overlay' or share data with any other column in the index. If you do have an ELSE clause, it will be ignored.

A group to be EXPANDED may contain 'nested' groups as well as items. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS index, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDED group; this is checked on encoding. This means that the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMs and/or GROUPs contained by the GROUP specified, or, failing that, from their dictionary names. The length of any column-name may be no longer than 16 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

### **Defining Free Space**

Use a PCTFREE clause to define the percentage of free space you want to be reserved in the index for later updates and insertions. The syntax of the clause is as follows:

```
>--- PCTFREE integer --->
```

where *integer* is an integer in the range 0 to 99, being the percentage of the total space of the index which you want to reserve. The clause is checked on encoding to ensure that the value specified is within the permitted range.

Example: SQL-INDEX Definition and SQL Generation

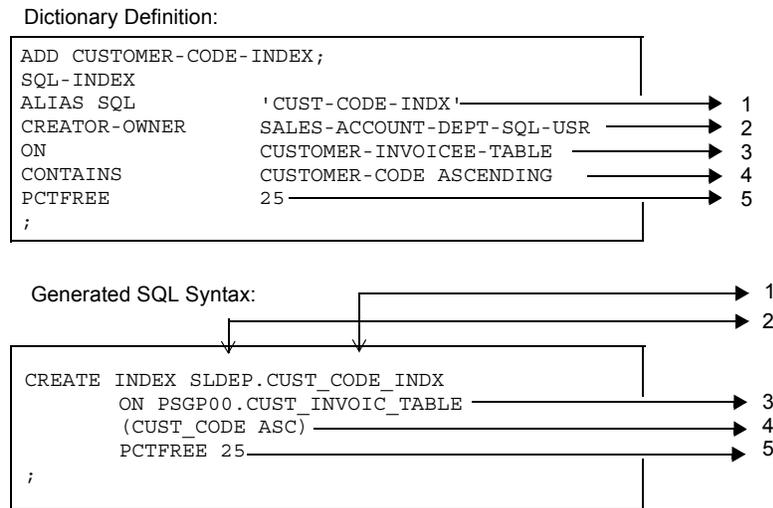
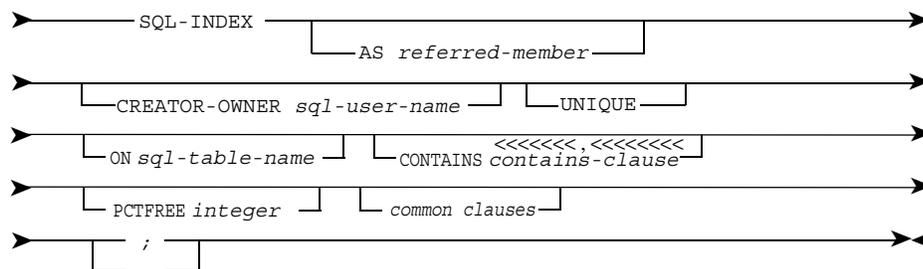


Figure 35 SQL-INDEX Definition and SQL Generation

1. The unqualified SQL/DS 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-SQL-USR) is used to fully qualify the index name. This is the SQL/DS object name for the index.
3. The dictionary member name of the table to be indexed is CUSTOMER-INVOICE-TABLE which is converted by the Name Reduction Process to an external name of CUST\_INVOIC\_TABLE. It is qualified by the table owner's Authorization ID, which is derived from the SQL ALIAS of the SQL-USER member which represents the Authorization ID (and which is defined in the CREATOR-OWNER clause of the table).
4. The CONTAINS clause is converted to an SQL column-specification by using the SQL ALIAS of member CUSTOMER-CODE and adding the SQL/DS abbreviation ASC for ASCENDING.
5. The PCTFREE parameter is taken directly from the dictionary definition.

Syntax of the SQL-INDEX Member Type



where:

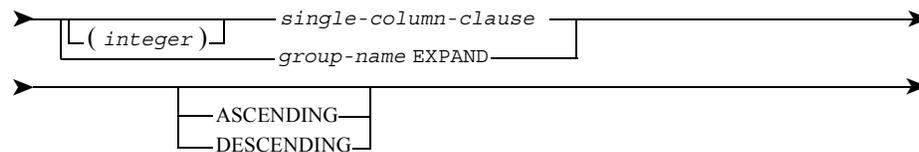
*referred-member* is the name of an SQL-INDEX dictionary member

*sql-user-name* is the name of an SQL-USER dictionary member

*sql-table-name* is the name of an SQL-TABLE dictionary member

*integer* is an integer in the range 0 to 99.

*contains-clause*:



where:

*integer* is the number of columns in a 'column set'

*group-name* is the name of a GROUP dictionary member.

*single-column-clause*:



where:

*item-name* is the name of an ITEM dictionary member

*group-name* is the name of a GROUP dictionary member

*local-name* consists of no more than 18 characters.

## SQL LABEL

Refer to "SQL COMMENT and SQL LABEL" on page 180 for details of the SQL LABEL command.

## SQL LIST CYCLES

Use the SQL LIST CYCLES command to identify the cycles found in the SQL design present in the Workbench Design Area (WBDA) and to list the tables which appear within each cycle.

### Use

A cycle can be described as 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

```
SQL LIST CYCLES ;
```

To list the tables in each cycle alphanumerically, enter:

```
SQL 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 SQL 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 SQL design, the command causes the SQL design to be generated before producing the list.

For each SQL 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 the following diagram, an example is pictured of a cycle with its path of tables and connecting relationships:

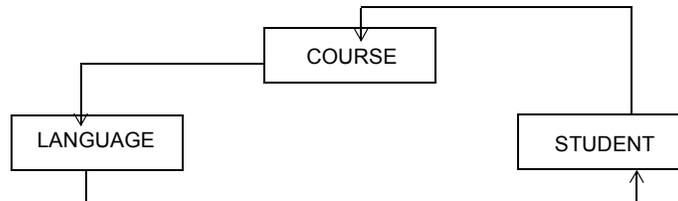


Figure 36 A Cycle and its Path of Tables and Relationships

Refer to "Output from the SQL LIST CYCLES Command" on page 74 for details of the SQL LIST CYCLES command output.

Refer to "Introduction to Referential Structures and Cycles" on page 27 for further discussion of cycles and how they can affect design decisions.

### Syntax of the SQL LIST CYCLES Command

```
SQL LIST CYCLES [ALPHABETICALLY] ;
```

### SQL LIST TABLES

Use the SQL LIST TABLES command to produce a list of all or some of the tables appearing in the SQL design generated in the Workbench Design Area (WBDA).

**Use**

To list all the tables in the SQL design in order of WBDA number, enter:

```
SQL LIST TABLES ;
```

To list all tables alphanumerically, enter:

```
SQL 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 SQL 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:

```
SQL LIST TABLES selection ;
```

where *selection* is one or more of the following 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:

```
SQL LIST TABLES ALPHABETICALLY selection ;
```

where *selection* is defined as above.

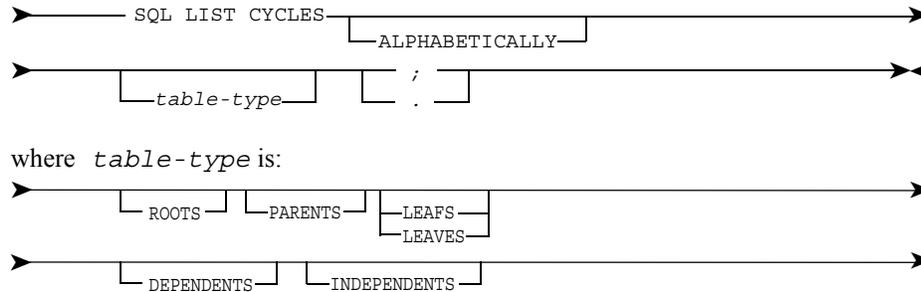
The command can be executed only if the WBDA contains normalized data. Otherwise, the command is terminated and a message of explanation is output. If the WBDA contains normalized data but no SQL design, the command causes the SQL design to be generated before producing the list.

For each SQL 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 "Description of the SQL LIST TABLES Output" on page 73 for a description of the various table types.

Refer to "Output from the SQL LIST TABLES Command" on page 73 for details of the SQL LIST TABLES command output.

### Syntax of the SQL LIST TABLES Command



### SQL PLOT CLUSTER

Use this command to produce an SQL Cluster Plot of all or some of the table in the SQL design.

Refer to "Syntax of the SQL PLOT CLUSTER Command" on page 202 for the syntax of the SQL PLOT CLUSTER command.

### Introduction to the SQL PLOT CLUSTER Command

Use the SQL PLOT CLUSTER command to produce a SQL Cluster Plot of all or some of the tables in the SQL 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 SQL design. When all the clusters have been displayed, the SQL Design Relationship Matrix is output. This is a two-dimensional table which summarizes all of the relationships holding between the tables of the SQL 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 SQL design, this command causes the SQL design to be generated and then produces the plot.

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 an SQL Cluster Plot displaying every table in the Workbench Design Area (WBDA), enter:

```
SQL PLOT CLUSTER ALL ;
```

This displays the tables in order of WBDA number.

To display all the tables alphanumerically, enter:

```
SQL 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 SQL Cluster Plot displaying tables selected by name, enter:

```
SQL 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 are displayed in the order listed unless the keyword ALPHABETICALLY also is specified in the command.

To display the tables in alphanumeric order of table name, enter:

```
SQL PLOT CLUSTER NAMES name-list ALPHABETICALLY ;
```

For example:

```
SQL PLOT CLUSTER NAMES DEPARTMENT, OFFICE, EMPLOYEE ALPHABETICALLY ;
```

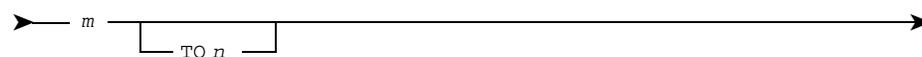
### Displaying Tables Selected by Number

This is the only way to select tables in the Workbench Design Area (WBDA) which have not yet been named.

To produce a SQL Cluster Plot of tables selected by their WBDA number, enter:

```
SQL 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:

```
SQL 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:

```
SQL 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:

```
SQL PLOT CLUSTER selection USING FORMAT format-member ;
```

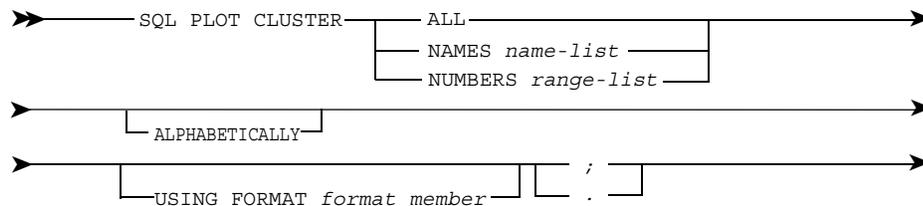
where:

*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 the following:

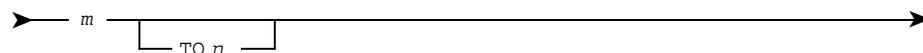
- ALL
- NAMES *name-list*
- NUMBERS *range-list*.

### Syntax of the SQL PLOT CLUSTER Command



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



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.

## SQL PLOT REFERENTIAL-STRUCTURES

Use this command to produce an SQL Referential Structures Plot of all or a single one of the referential structures in the SQL design.

Refer to "Syntax of the SQL PLOT REFERENTIAL-STRUCTURES Command" on page 207 for the syntax of the PLOT REFERENTIAL-STRUCTURES command.

### Introduction to the SQL PLOT REFERENTIAL-STRUCTURES Command

Use the SQL PLOT REFERENTIAL-STRUCTURES command to produce the SQL Referential Structures Plot, a consolidated overview display of one or all of the referential structures in the SQL 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 SQL design, the command causes the SQL 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.

The following diagram illustrates a referential structure in the case of the Department Model example:

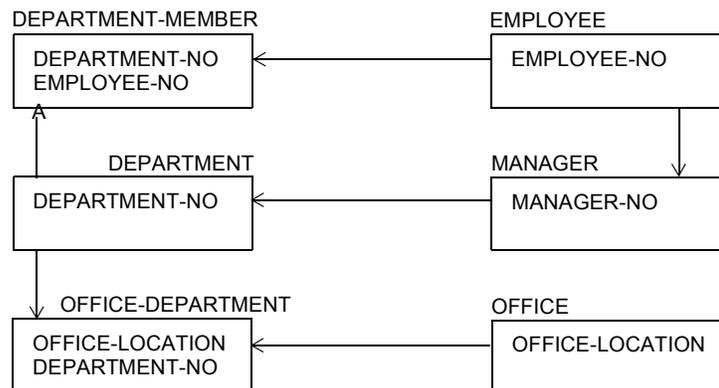


Figure 37 Department Model: A Referential Structure

For each referential structure displayed in a SQL 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*.

You can display all of the referential structures of the SQL 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 27 for further discussion of referential structures.

### **Displaying All Referential Structures**

Use the ALL keyword in the SQL PLOT REFERENTIAL-STRUCTURES command to display all of the referential structures of the SQL design in the SQL 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 SQL design is displayed in the SQL 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:

```
SQL PLOT REFERENTIAL-STRUCTURES ALL ;
```

without specifying either PARENTS or DEPENDENTS.

Then, for each referential structure in the SQL 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 SQL 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 or more cycles instead of root parents, then ASG-DesignManager selects as the next seed the lowest numbered (non-independent) table remaining in the SQL 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 SQL 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:

```
SQL 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 SQL 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:

```
SQL 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 SQL 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 SQL 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 SQL 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 SQL PLOT REFERENTIAL-STRUCTURES command to display all or part of a single referential structure (or a single independent table) from the SQL 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 SQL 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:

```
SQL PLOT REFERENTIAL-STRUCTURES SEND selection ;
```

without specifying either PARENTS or DEPENDENTS.

*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 SQL 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:

```
SQL PLOT REFERENTIAL-STRUCTURES SEED selection DEPENDENTS ;
```

where *selection* is defined as above.

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.)

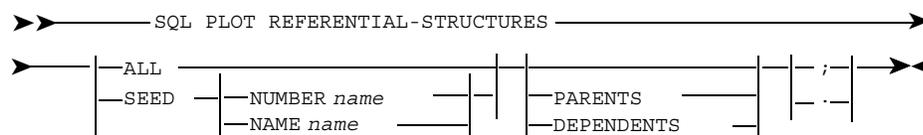
To display only parent tables and relationships (following the seed), enter:

```
SQL PLOT REFERENTIAL-STRUCTURES SEED selection PARENTS ;
```

where *selection* is defined as above.

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.)

### Syntax of the SQL PLOT REFERENTIAL-STRUCTURES Command



where:

*number* is a table number in the Workbench Design Area

*name* is a table name.

## **SQL POPULATE**

Use the SQL POPULATE command to populate the dictionary with SQL-TABLE, SQL-INDEX, and SQL-VIEW members, generated from the SQL design, and optionally to produce a report of the generated members.

Refer to "Syntax of the SQL POPULATE Command" on page 217 for the syntax of the SQL POPULATE command.

### *Introduction to the SQL POPULATE Command*

The SQL POPULATE command generates dictionary member definitions and populates the dictionary with them. A report of the generated member definitions is automatically output.

The SQL POPULATE command generates dictionary member definitions for one or more of the following member types, from selected tables of the SQL design in the Workbench Design Area (WBDA):

- SQL-TABLE
- SQL-INDEX
- SQL-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 SQL 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 SQL POPULATE command will automatically generate the SQL design if one has not already been generated. Any unnamed tables will be ignored by the command.

Although the content of a generated SQL 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 SQL object, SQL/DS will assign default values to all the remaining clauses.)

Each member definition is preceded by an ADD command and followed by a terminator.

SQL POPULATE automatically generates primary key keywords and foreign key clauses in SQL-TABLE definitions to support referential integrity, unless you use the NO-RI option to suppress them. In addition, the command enables you to assign SQL tables to specific dbspaces (the DBSPACE option).

The command also allows you to associate a dictionary SQL-USER member (via the CREATOR-OWNER clause) with the SQL 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 SQL 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 SQL POPULATE with a NOPRINT command you can stop any output being printed.

Refer to the *ASG-ControlManager User's Guide* for details of the NOPRINT command.

### **Generating and Populating SQL-TABLE Members**

Use the keyword TABLES to generate and populate the dictionary with SQL-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 SQL-TABLE dictionary member.

To populate the dictionary with SQL-TABLE members, enter:

```
SQL 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 SQL-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.

The following clauses are generated for each SQL-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 SQL-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.
- 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 SQL PREVIEW command, a CREATOR-OWNER clause is included specifying a dictionary SQL-USER member as the creator or owner of the SQL table
- If specified in the SQL PREVIEW command, an IN IN dbspace-name clause is included, specifying the name of a dictionary SQL-DBSPACE member.

Refer to "Generated SQL-TABLE Definition" on page 76 for an example of SQL-TABLE definition syntax generated by the SQL POPULATE command.

### **Suppressing Support for Referential Integrity**

To specify that the SQL-TABLE members must not contain clauses supporting referential integrity (RI), enter:

```
SQL 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 Dbspaces**

To generate a reference to a SQL-DBSPACE dictionary member in each generated SQL-TABLE dictionary definition, enter:

```
SQL POPULATE TABLES DBSPACES 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 SQL-DBSPACE member is constructed from the name of the table concatenated with the suffix '-DBSP', 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 SQL-TABLE member defined, enter:

```
SQL POPULATE TABLES DBSPACES 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:

```
SQL POPULATE TABLES DBSPACES PREFIX 'string' selection ;
```

or

```
SQL POPULATE TABLES DBSPACES SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

### **Generating and Populating SQL-INDEX Members**

To populate the dictionary with a SQL-INDEX member representing a primary index for each selected table, enter:

```
SQL 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 SQL-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 SQL-INDEX name from the name of the table concatenated with a specified prefix or suffix, enter:

```
SQL POPULATE INDEXES PREFIX 'string' selection ;
```

or

```
SQL POPULATE INDEXES SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

The following clauses are generated for each SQL-INDEX member:

- The CONTAINS clause of the SQL-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 SQL-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' sub-clause
- If specified in the SQL POPULATE command, a CREATOR-OWNER clause is also included which names a dictionary SQL-USER member as the creator or owner of the SQL

Refer to "Generated SQL-INDEX Definition" on page 77 for the syntax of a SQL-INDEX definition generated by the SQL POPULATE command.

### **Generating and Populating SQL-VIEW Members**

To populate the dictionary with a SQL-VIEW member for each selected table, enter:

```
SQL 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 SQL-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 SQL-VIEW name from the name of the table concatenated with a specified prefix or suffix, enter:

```
SQL POPULATE VIEWS PREFIX 'string' selection ;
```

or

```
SQL POPULATE VIEWS SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

The following clauses are generated for each SQL-VIEW member definition:

- The CONTAINS clause of the SQL-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 SQL-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' sub-clause
- If specified in the SQL PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary SQL-USER member as the creator or owner of the SQL view.

Refer to "Generated SQL-VIEW Definition" on page 78 for the syntax of a SQL-VIEW definition generated by the SQL POPULATE command.

### **Generating References to a SQL User**

To specify that a selection of members generated from the Workbench Design Area (WBDA) belongs to a particular SQL-USER member, enter:

```
SQL POPULATE member-type-selection CREATOR-OWNER sql-user  
selection ;
```

where:

*member-type-selection* is one or more of the TABLES, INDEXES, and VIEWS clauses

*sql-user* is the name of a dictionary SQL-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 SQL-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 SQL members generated by this command, enter:

```
SQL POPULATE 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 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 SQL POPULATE command.

Refer to "Example of Generated SYSTEM Dictionary Member" on page 80 for the syntax of the SYSTEM member generated by the SQL POPULATE command.

### Selecting Tables in the Workbench Design Area

When issuing the SQL 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:

```
SQL 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:

```
SQL POPULATE member-type-selection NAME 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.

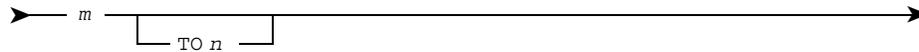
To specify the tables by WBDA number, enter:

```
SQL 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:

```
SQL POPULATE member-type-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 SQL dictionary definitions in a format tailored to your requirements, enter:

```
SQL POPULATE member-type-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 SQL dictionary definitions compatible with any User Defined Syntax structure you may have implemented
- Generate dictionary 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 SQL POPULATE Command Options**

You can generate and populate the dictionary with SQL member definitions for any combination of the SQL-TABLE, SQL-INDEX, and SQL-VIEW member types in one SQL 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 SQL 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 SQL member definitions.

### **Examples of the SQL POPULATE Command**

This section shows examples of the combinations of options available in the SQL POPULATE command.

To generate and populate a SQL-TABLE member:

- For a table named DEPARTMENT,
- With no clauses for referential integrity (RI), and
- Belonging to a system called SQL-SYSTEM-TEST,

enter:

```
SQL POPULATE TABLES NO-RI AS-SYSTEM SQL-SYSTEM-TEST NAMES  
DEPARTMENT ;
```

To generate and populate SQL-TABLE and SQL-INDEX members:

- For all the tables in the WBDA,
- Including clauses to support RI,
- Constructing each SQL-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:

```
SQL POPULATE TABLES INDEXES SUFFIX 'TEST' ALL ALPHABETICALLY ;
```

To generate and populate SQL-INDEX and SQL-VIEW members:

- For tables in the WBDA selected by WBDA number,
- With each SQL-INDEX definition name constructed from the table name concatenated with the default suffix 'IND',
- Constructing each SQL-VIEW definition name from the table name concatenated with the prefix 'TEST', and
- Specifying that all the generated SQL-INDEX and SQL-VIEW definitions must reference a SQL-USER member named USER1,

enter:

```
SQL POPULATE INDEXES VIEWS PREFIX 'TEST' CREATOR-OWNER
                                USER1 NUMBERS 1 To 3, 5 ;
```

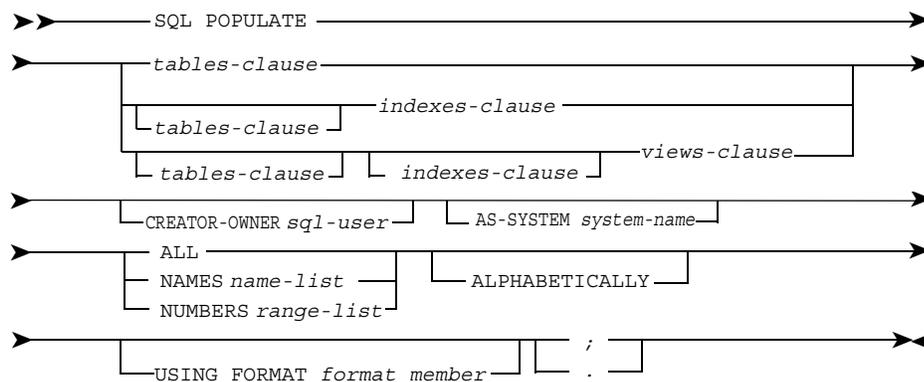
To generate and populate SQL-TABLE and SQL-VIEW members:

- For all tables in the WBDA,
- Suppressing clauses to support RI,
- Specifying that all the generated SQL-TABLE members must reference a SQL-DBSPACE member named DEP-DBSP, and
- Formatting the output according to a format definition named FMT-REPL,

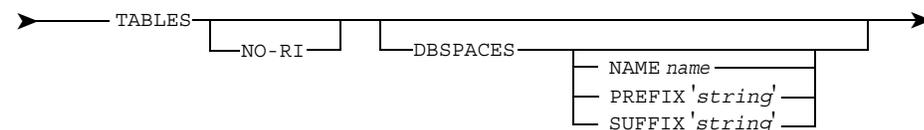
enter:

```
SQL POPULATE TABLES NO-RI DBSPACES NAME DEP-DBSP VIEWS ALL
                                USING FORMAT FMT-REPL ;
```

### Syntax of the SQL POPULATE Command



where *tables-clause* is:

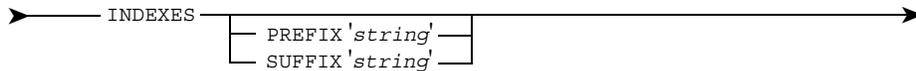


where:

*name* is an alphanumeric string of up to 32 characters which should conform to the rules for a valid Manager Products dictionary member name

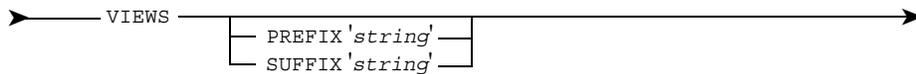
*string* is an alphanumeric string of up to 31 characters which should conform to the rules for a valid Manager Products dictionary member name.

*indexes-clause* is:



where *string* is defined as above.

*views-clause* is:



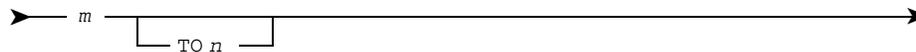
where *string* is defined as above.

*sql-user* is an alphanumeric string of up to 32 characters, which should conform to the rules for a valid Manager Products dictionary member name

*system-name* is an alphanumeric string of up to 32 characters, which should conform to the rules for a valid Manager Products dictionary 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:



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.

## SQL PREVIEW

Use the SQL PREVIEW command to generate and report dictionary about SQL/DS member definitions from SQL tables, indexes, and views in the SQL design.

Refer to "Syntax of the SQL PREVIEW Command" on page 228 for the syntax of the SQL PREVIEW command.

### *Introduction to the SQL PREVIEW Command*

The SQL PREVIEW command generates dictionary member definitions for one or more of the following member types, from selected tables of the SQL design in the Workbench Design Area (WBDA):

- SQL-TABLE
- SQL-INDEX
- SQL-VIEW.

Each time you issue the command, a SYSTEM member also can be generated, containing a list of all the SQL 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 SQL PREVIEW command will automatically generate the SQL 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 SQL POPULATE command.

Although the content of a generated SQL 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 SQL object, SQL/DS will assign default values to all the remaining clauses.)

Each member definition is preceded by an ADD command and followed by a terminator.

SQL PREVIEW automatically generates primary key keywords and foreign key clauses in SQL-TABLE definitions to support referential integrity, unless you use the NO-RI option to suppress them. In addition, the command enables you to assign SQL tables to specific dbspaces (the DBSPACE option).

The command also allows you to associate a dictionary SQL-USER member (via the CREATOR-OWNER clause) with the SQL 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 SQL 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 SQL-TABLE Definitions**

Use the keyword `TABLES` to generate and preview a SQL-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 SQL-TABLE member.

To generate and preview SQL-TABLE definitions, enter:

```
SQL 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 SQL-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.

The following clauses are generated for each SQL-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 SQL-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.
- 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 SQL PREVIEW command, a CREATOR-OWNER clause is included specifying a dictionary SQL-USER member as the creator or owner of the SQL table
- If specified in the SQL PREVIEW command, an IN dbspace-name clause is included, specifying the name of a dictionary SQL-DBSPACE member.

Refer to "Generated SQL-TABLE Definition" on page 76 for the syntax used for a dictionary SQL-TABLE definition generated by the SQL PREVIEW command.

### Suppressing Support for Referential Integrity

To specify that the SQL-TABLE definitions must not contain clauses supporting referential integrity (RI), enter:

```
SQL 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 Dbspaces

To generate a reference to a SQL-DBSPACE dictionary member in each generated SQL-TABLE dictionary definition, enter:

```
SQL PREVIEW TABLES DBSPACES 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 SQL-DBSPACE member is constructed from the name of the table concatenated with the suffix '-DBSP', 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 SQL-TABLE member defined, enter:

```
SQL PREVIEW TABLES DBSPACES 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:

```
SQL PREVIEW TABLES DBSPACES PREFIX 'string' selection ;
```

or

```
SQL PREVIEW TABLES DBSPACES SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

### **Generating and Previewing SQL-INDEX Definitions**

To specify that you want a SQL-INDEX definition, representing a primary index, to be generated for each selected table, enter:

```
SQL 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 SQL-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 SQL-INDEX name from the name of the table concatenated with a specified prefix or suffix, enter:

```
SQL PREVIEW INDEXES PREFIX 'string' selection ;
```

or

```
SQL PREVIEW INDEXES SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

The following clauses are generated for each SQL-INDEX definition:

- The CONTAINS clause of the SQL-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 SQL-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' sub-clause
- If specified in the SQL PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary SQL-USER member as the creator or owner of the SQL index.

Refer to "Generated SQL-INDEX Definition" on page 77 for the syntax of a SQL-INDEX definition generated by the SQL PREVIEW command.

### Generating and Previewing SQL-VIEW Definitions

To specify that you want a SQL-VIEW definition to be generated for each selected table, enter:

```
SQL 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 SQL-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 SQL-VIEW definition's name from the table name concatenated with a specified prefix or suffix, enter:

```
SQL PREVIEW VIEWS PREFIX 'string' selection ;
```

or

```
SQL PREVIEW VIEWS SUFFIX 'string' selection ;
```

where:

*string* is a valid dictionary string of up to 31 characters

*selection* is defined as above.

The following clauses are generated for each SQL-VIEW definition:

- The CONTAINS clause of the SQL-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 SQL-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' sub-clause
- If specified in the SQL PREVIEW command, a CREATOR-OWNER clause is also included which names a dictionary SQL-USER member as the creator or owner of the SQL view.

Refer to "Generated SQL-VIEW Definition" on page 78 for the syntax of a SQL-VIEW definition generated by the SQL PREVIEW command.

### **Generating References to a SQL User**

To specify that a selection of members generated from the Workbench Design Area (WBDA) belongs to a particular SQL-USER member, enter:

```
SQL PREVIEW member-type-selection CREATOR-OWNER sql-user  
selection ;
```

where:

*member-type-selection* is one or more of the TABLES, INDEXES, and VIEWS clauses

*sql-user* is the name of a dictionary SQL-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 SQL dictionary definitions. If the SQL-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 SQL definitions generated by this command, enter:

```
SQL 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 SQL PREVIEW command.

Refer to Relational Technology Support: SQL/DS manual 3.4.4.1 for the syntax of the SYSTEM definition generated by the SQL PREVIEW command.

### Selecting Tables in the Workbench Design Area

When issuing the SQL 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:

```
SQL 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:

```
SQL 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.

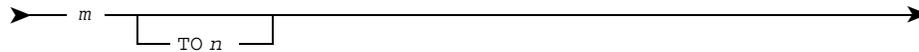
To specify the tables by WBDA number, enter:

```
SQL 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 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:

```
SQL 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 SQL dictionary definitions in a format tailored to your requirements, enter:

```
SQL 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 SQL dictionary definitions compatible with any User Defined Syntax structure you may have implemented
- Generate dictionary 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 SQL PREVIEW Command Options

You can generate dictionary definitions for any combination of the SQL-TABLE, SQL-INDEX, and SQL-VIEW member types in one SQL 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
- 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 SQL member definitions.

### Examples of the SQL PREVIEW Command

This panel shows examples of the combinations of options available in the SQL PREVIEW command.

To generate and preview a SQL-TABLE definition:

- For a Workbench Design Area (WBDA) table named DEPARTMENT,
- With no clauses for referential integrity (RI), and
- Belonging to a system called SQL-SYSTEM-TEST,

enter:

```
SQL PREVIEW TABLES NO-RI AS-SYSTEM SQL-SYSTEM-TEST NAMES
DEPARTMENT ;
```

To generate and preview SQL-TABLE and SQL-INDEX definitions:

- For all the tables in the WBDA,
- Including clauses to support RI,
- Constructing each SQL-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:

```
SQL PREVIEW TABLES INDEXES SUFFIX 'TEST' ALL ALPHABETICALLY ;
```

To generate and preview SQL-INDEX and SQL-VIEW definitions:

- For tables in the WBDA selected by WBDA number,
- With each SQL-INDEX definition name constructed from the table name concatenated with the default suffix 'IND',
- Constructing each SQL-VIEW definition name from the table name concatenated with the prefix 'TEST', and
- Specifying that all the generated SQL-INDEX and SQL-VIEW definitions must reference a SQL-USER member named USER1,

enter:

```
SQL PREVIEW INDEXES VIEWS PREFIX 'TEST' CREATOR-OWNER USER1
NUMBERS 1 TO 3, 5 ;
```

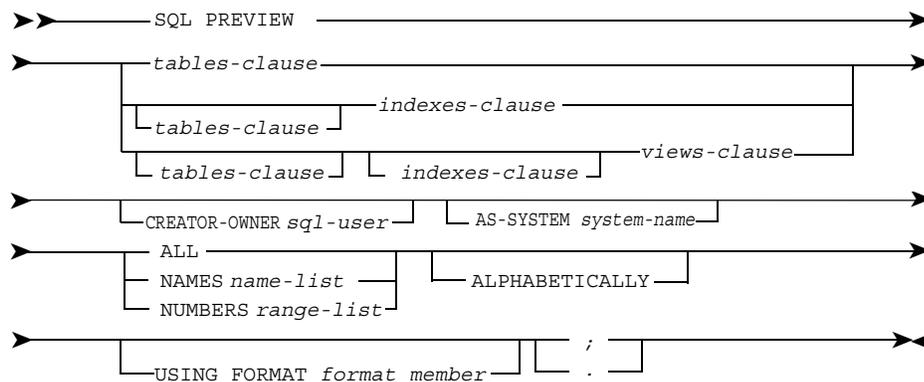
To generate and preview SQL-TABLE and SQL-VIEW definitions:

- For all tables in the WBDA,
- Suppressing clauses to support RI,
- Specifying that all the generated SQL-TABLE definitions must reference a SQL-DBSPACE definition named DEP-DBSP, and
- Formatting the output according to a format definition named FMT-REPL,

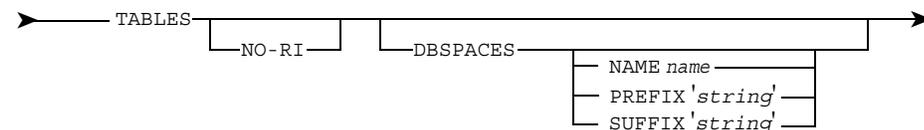
enter:

```
SQL PREVIEW TABLES NO-RI DBSPACES NAME DEP-DBSP VIEWS ALL USING
FORMAT FMT-REPL ;
```

### Syntax of the SQL PREVIEW Command



where *tables-clause* is:

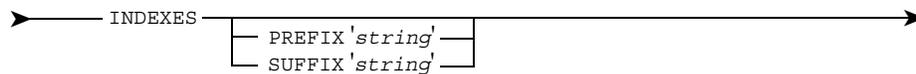


where:

*name* is an alphanumeric string of up to 32 characters which should conform to the rules for a valid Manager Products dictionary member name

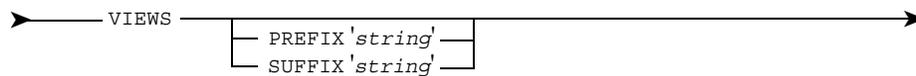
*string* is an alphanumeric string of up to 31 characters which should conform to the rules for a valid Manager Products dictionary member name.

*indexes-clause* is:



where *string* is defined as above.

*views-clause* is:



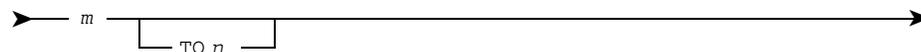
where *string* is defined as above.

*sql-user* is an alphanumeric string of up to 32 characters, which should conform to the rules for a valid Manager Products dictionary member name

*system-name* is an alphanumeric string of up to 32 characters, which should conform to the rules for a valid Manager Products dictionary 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:



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.

## SQL-PRIVILEGE

SQL privileges are defined in the dictionary as SQL-PRIVILEGE members.

Refer to "Syntax of the SQL-PRIVILEGE Member Type" on page 235 for the syntax of the SQL-PRIVILEGE member type.

### Introduction to SQL-PRIVILEGE

To document an SQL/DS privilege in the dictionary use the SQL-PRIVILEGE member type. To define the dictionary member type enter SQL-PRIVILEGE at the start of your member definition statement.

In the SQL/DS environment you can grant privileges to specific users to allow them to access particular tables, views, and programs. You can also grant system authorities to users (in a SYSTEM privilege).

You can record all of these types of privilege in the dictionary as members of the type SQL-PRIVILEGE, the distinction between them being made by an appropriate keyword in the data definition. You may record only one type of privilege in one member.

Table and view privileges give specified users access to particular SQL/DS objects. For example, an UPDATE privilege on a table gives a user the ability to perform UPDATES on a particular, named, table (defined in the ON clause).

Program privileges allow specified users to run particular programs, and system authority privileges allow you to grant specified users different levels of administrative authority over the SQL/DS environment.

You can record the grantors and recipients of privileges (in GRANTOR and TO clauses respectively).

Table, view, and program privileges can be granted with or without the GRANT option (by including or excluding the WITH-GRANT-OPTION keyword). A privilege with a GRANT option is transferable, that is, the recipient may pass on the privilege to another user.

Clauses may be declared in any order.

The privilege-type clause and the TO clause must be present for the successful generation of an SQL GRANT statement.

The definitions recorded in the dictionary are used to generate SQL GRANT and SQL REVOKE statements. They can also be interrogated using dictionary interrogation commands to provide database administrators with the ability to analyze the dictionary model of the SQL/DS security system.

When you generate an SQL REVOKE statement from an SQL-PRIVILEGE member, you must consider carefully whether to remove the member from the dictionary. You may need to grant the privilege again, in which case you should retain the member in the dictionary. If you do this, the privileges documented in the dictionary will reflect your long-term security arrangements and can be used whenever necessary to GRANT and REVOKE SQL privileges.

### **The AS Clause**

Refer to "Defining an AS Clause" on page 277 for details of the AS clause.

### **Defining the Grantor of an SQL/DS Privilege**

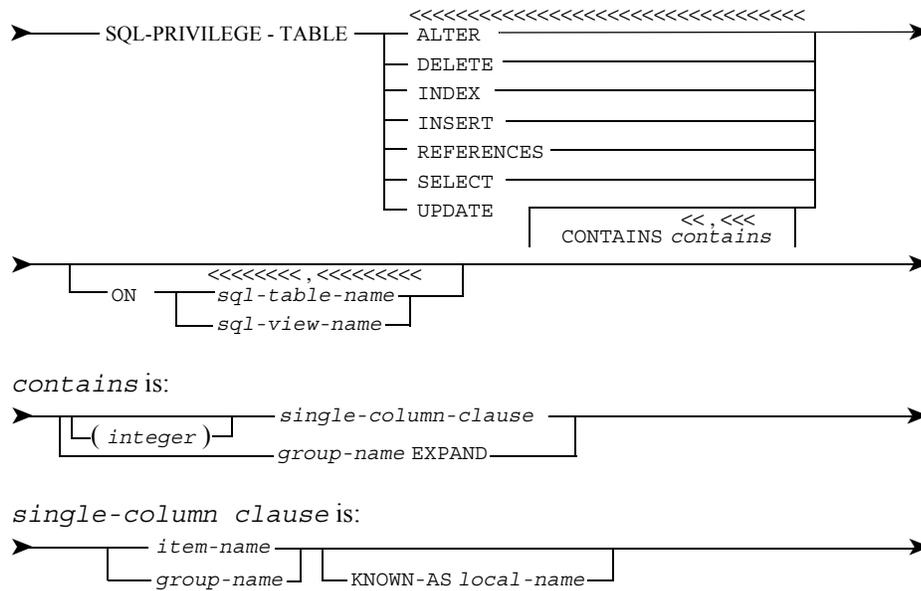
Use a GRANTOR clause to define the user who is granting the privilege. The syntax of the clause is as follows:

➤ GRANTOR *sql-user* ➤

where *sql-user* is the name of a dictionary member of the type SQL-USER, which represents the Authorization ID of the user who is granting the privilege. (Typically, the grantor will be a database administrator for a project.) The clause is checked on encoding to ensure that the member specified is of the correct type. This clause is used for documentation purposes only; it is optional and does not affect the generation of an SQL GRANT statement.

**Defining Specific Privileges on Tables and Views**

Use the keyword TABLE to grant privileges on tables and views. The syntax is as follows:



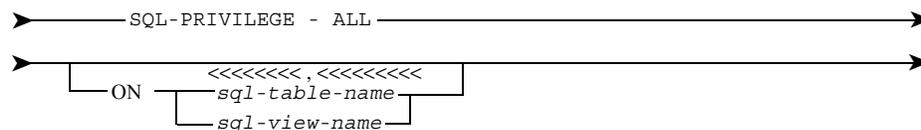
The keywords are the same as those used by SQL/DS and they have the same meanings. Use the CONTAINS clause to define the column or columns which the UPDATE privilege applies to. The column(s) thus specified must be defined in the same way as they are in the corresponding clauses of the SQL-TABLE or SQL-VIEW member, which represents the SQL/DS object to which the privilege applies, and they must conform to the same requirements.

where *sql-table-name* and *sql-view-name* are dictionary members of the appropriate type, which represent the object(s) on which the privilege operates.

The clause is checked on encoding to ensure that the privilege specified is a valid one.

**Defining ALL Privileges on Tables and Views**

Use the keyword ALL to grant all possible privileges a user may have on a table or view. The syntax is as follows:



where *sql-table-name* and *sql-view-name* are dictionary members of the appropriate type, which represent the object(s) on which the privilege operates.

### Defining SQL/DS PROGRAM Privileges

Use the PROGRAM ON clause to grant program privileges. The syntax is as follows:

➤ SQL-PRIVILEGE - PROGRAM ON <<<<<<, <<<<< program-name ➤

where *program-name* is the name of the PROGRAM which the recipient of the privilege is permitted to run. The PROGRAM must be a dictionary member and it must have a valid CREATOR-OWNER clause. The SQL-USER specified in the CREATOR-OWNER clause is the original grantor of this PROGRAM privilege.

The clause is checked on encoding to ensure that the privilege specified is a valid one.

### Defining the Recipient of SQL/DS Privileges on Tables, Views, and Programs

Use a TO clause to define the user to whom a privilege on a table, view, or program is being granted. The syntax is as follows:

➤ TO <<<<, <<<<<< sql-user-name WITH-GRANT-OPTION ➤

where *sql-user-name* is the name of a member of the type SQL-USER, which represents the Authorization ID of the user to whom the privilege is being granted. The clause is checked on encoding to ensure that the member specified is of the correct type.

Use the optional WITH-GRANT-OPTION keyword to specify that the recipient(s) of the privilege may transfer it to another user.

This clause must be present for the successful generation of an SQL statement.

### Defining SQL/DS SYSTEM Privileges and their Recipients

Use the SYSTEM clause to grant system privileges. The syntax is as follows:

➤ SYSTEM CONNECT RESOURCE DBA SCHEDULE subsystem-id TO <<<<, <<<<< sql-user-name IDENTIFIED-BY <<<, <<< password ➤

where:

*sql-user-name* is the name of a dictionary member of the type SQL-USER, which represents the Authorization ID of the user who is being granted the privilege specified. The clause is checked on encoding to ensure that the member specified is of the correct type.

*subsystem-id* is the ID of the CICS subsystem running under the VSE guest

*password* is a password associated with the SQL/DS user specified, and every user must have an associated password. The subsystem-id may have only one associated password. The password may consist of no more than eight characters; this is checked on encoding. The IDENTIFIED-BY clause may be included if you need to change a password, or specify a new one. The clause must be present for successful generation when a privilege is being granted to a subsystem-id.

### Examples of SQL-PRIVILEGE Definitions and SQL Generation

#### Example 1: SQL-PRIVILEGE TABLE Definition and SQL Generation

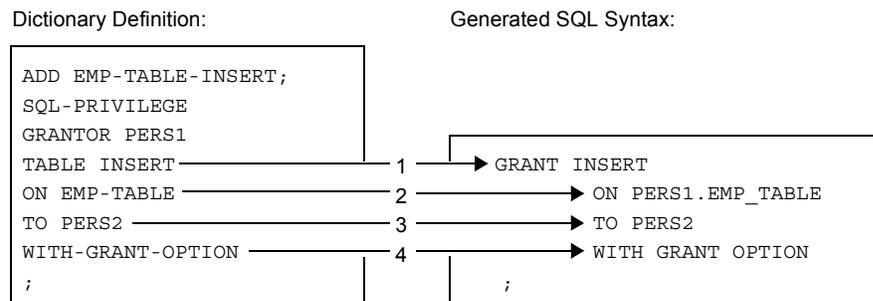


Figure 38 SQL-PRIVILEGE TABLE Definition and SQL Generation

1. The type of privilege being granted is taken directly from the data definition statement.
2. The SQL/DS name for the table on which the privilege is being granted is derived from the dictionary name of the SQL-TABLE member specified in the dictionary definition, qualified by the CREATOR-OWNER of the table, as specified in the SQL-TABLE member definition.
3. The SQL/DS name of the user to whom the privilege is being granted is taken directly from the dictionary definition.
4. The WITH GRANT OPTION is taken directly from the dictionary definition.

**Example 2: SQL-PRIVILEGE PROGRAM Definition and SQL Generation**

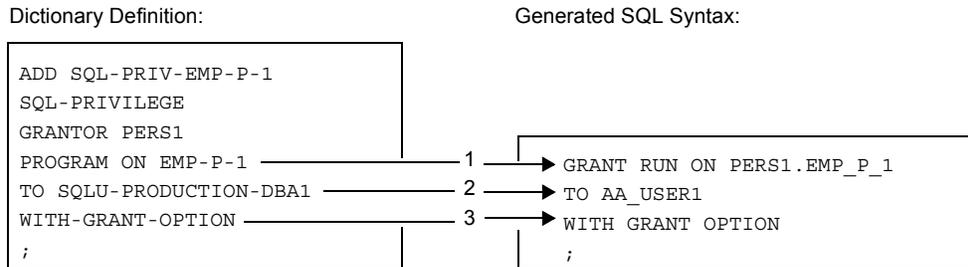


Figure 39 SQL-PRIVILEGE PROGRAM Definition and SQL Generation

1. The type of privilege being granted is taken directly from the member definition statement. The SQL/DS name of the program on which RUN privileges are being granted is the dictionary name of the program, qualified by the CREATOR-OWNER of the program.
2. The SQL/DS name of the user to whom the privilege is being granted is derived from the SQL ALIAS of the dictionary member, SQLU-PRODUCTION-DBA1, which represents the user.
3. The WITH GRANT OPTION is taken directly from the member definition.

**Example 3: SQL-PRIVILEGE SYSTEM Definition and SQL Generation**

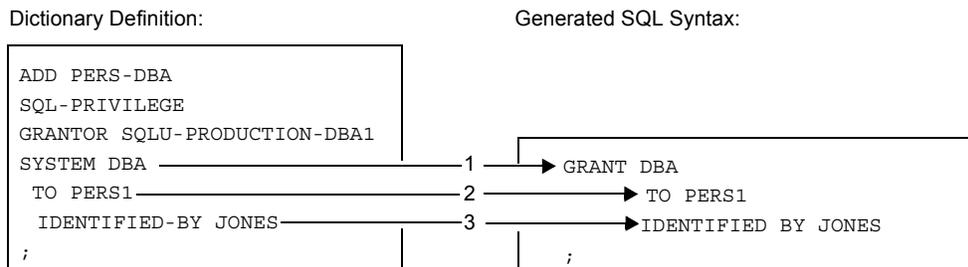
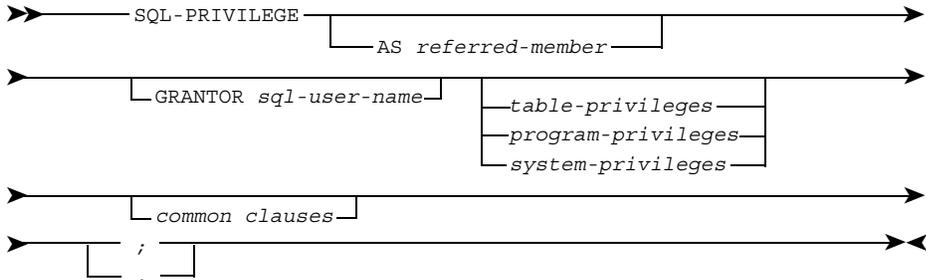


Figure 40 SQL-PRIVILEGE SYSTEM Definition and SQL Generation

1. The type of privilege being granted is taken directly from the member definition statement.
2. The Authorization ID of the recipient of the privilege is taken directly from the member definition statement.
3. The password associated with the Authorization ID is taken directly from the member definition statement.

Syntax of the SQL-PRIVILEGE Member Type

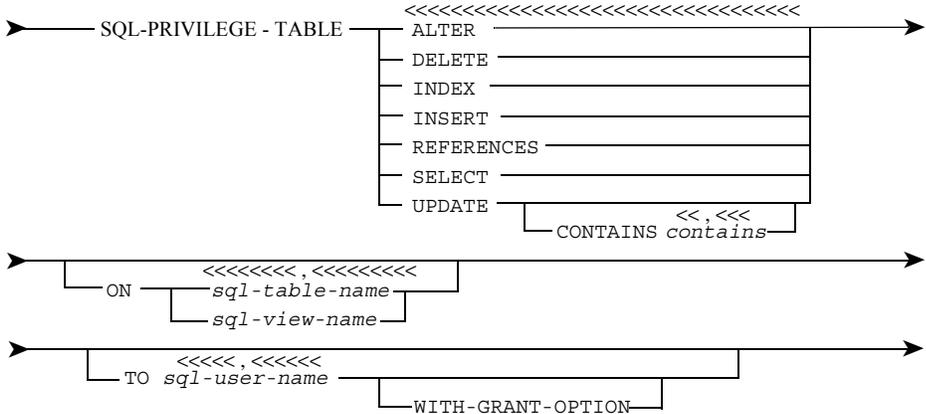


where:

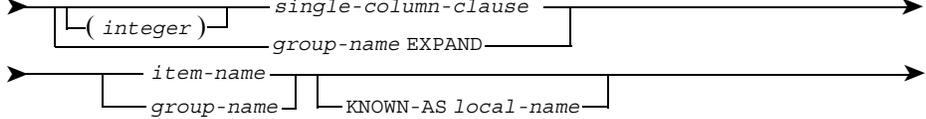
referred-member is the name of an SQL-PRIVILEGE dictionary member

sql-user-name is the name of an SQL-USER dictionary member.

table-privileges:



contains-clause:



where:

sql-table-name is the name of an SQL-TABLE dictionary member

sql-view-name is the name of an SQL-VIEW dictionary member

sql-user-name is the name of an SQL-USER dictionary member

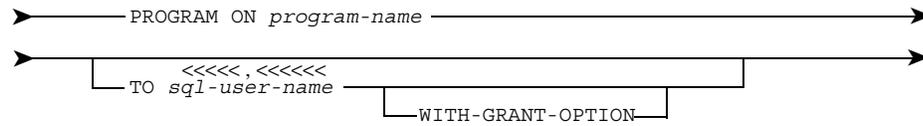
(integer) is the number of columns in a 'column set'

item-name is the name of an ITEM dictionary member

*group-name* is the name of a GROUP dictionary member

*local-name* consists of no more than eighteen characters.

*program-privilege:*

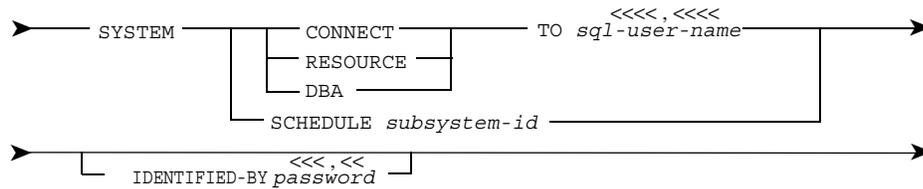


where:

*program-name* is the name of a PROGRAM dictionary member

*sql-user-name* is the name of an SQL-USER dictionary member.

*system-privileges:*



where:

*sql-user-name* is the name of an SQL-USER dictionary member

*subsystem-id* is the name of the subsystem ID of the CICS subsystem

*password* is an SQL Authorization ID password, consisting of no more than eight characters.

## SQL PRODUCE

Use the SQL PRODUCE command to generate a host language data structure or a table layout from the definition of an SQL-TABLE or SQL-VIEW member.

Refer to "Syntax of the SQL PRODUCE Command" on page 239 for the syntax of the SQL PRODUCE command.

### Introduction to the SQL PRODUCE Command

Use the SQL PRODUCE command to generate:

- A host language data structure, or
- A table layout

from the definition of a SQL-TABLE or SQL-VIEW member.

The output generated can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file the generated output in a USER-MEMBER you must specify an ONTO clause in the SQL PRODUCE command.

### **Generating a Host Language Data Structure**

To generate a host language data structure, enter:

```
SQL PRODUCE language FROM member-name ;
```

where:

*language* is ASSEMBLER (or ALC or BAL), PL1 (or PLI, PL/I or PL/1), or COBOL

*member-name* is an encoded SQL-TABLE or SQL-VIEW member.

Column variables in the host language data structure are generated from the definition of the ITEMS or GROUPs specified in the CONTAINS clause of the SQL-TABLE or SQL-VIEW member specified in the SQL PRODUCE command.

ASSEMBLER, PLI, or COBOL data types are generated for the column variables and correspond to the SQL/DS data type of the columns in the table or view.

The Systems Administrator can tailor the output of the SQL PRODUCE command so that:

- Host language indicator structures are generated
- External names are derived from aliases
- Internal diagnostic output is displayed
- Character field lengths are compatible with your SQL/DS installation settings for time and date.

The generated column and indicator structures can be referenced by application programs containing imbedded SQL syntax for data manipulation statements.

Refer to "Generating Column Data Types" on page 96 for details of generating data types for column variables.

### **Generating an SQL/DS Table Layout**

Use the SQL PRODUCE command to generate a table layout from the definition of a SQL-TABLE or SQL-VIEW member.

To generate a table layout, enter:

```
SQL PRODUCE TABLE-LAYOUT FROM member-name ;
```

where *member-name* is an encoded SQL-TABLE or SQL-VIEW member.

A table layout displays the column structure of a table or view by listing the SQL-TABLE, SQL-VIEW, GROUP, and ITEM members documenting the table or view, and its columns, in the dictionary.

Information such as the SQL/DS data type and length of columns, as well as if any of them are documented in the dictionary as being null or primary keys, is displayed in a convenient tabular format.

The DESCRIPTION clause of each of the members listed in the table layout is also displayed. DESCRIPTION clauses can contain 32767 delimited character strings and therefore enable you to describe SQL/DS objects in greater detail than that possible in SQL/DS comments or labels.

By printing the table layouts you can produce paper documentation displaying the structure and purpose of your tables and views.

The size of the table layouts you can generate is determined by the maximum number of lines of output that can be printed in any output buffer. The Systems Administrator can specify the maximum line limit with the SET OUTPUT-LINE-LIMIT command. Use the QUERY OUTPUT-LINE-LIMIT command to find out the current line limit. The maximum line limit does not restrict the size of table layouts generated by SQL PRODUCE commands entered in batch.

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

Refer to the *ASG-Manager Products Systems Administrator's Manual* for details of the SET OUTPUT-LINE-LIMIT command.

### **A Description of Table Layouts**

A table layout lists and describes the members documenting the columns in a table or view and the edit procedures, field procedures, and validation procedures of a DB2 table.

The entries under the heading PKEY specify whether a column is documented in the dictionary as being a primary key. The column is a primary key if Y is specified and is not a primary key if there is no entry.

The entries under the heading NULL specify whether a column is documented in the dictionary as being null. The column is null if N is specified and is not null if there is no entry, or, not null with default values if D is specified. D entries only apply to DB2 tables.

The entries under the heading NAME specify the names of the members documenting the table or view and their columns and edit, field, and validation procedures.

The entries under the heading TYPE specify the DB2 or SQL/DS data type of a column, or, identify an edit, field, or validation procedure. Data types are derived from the definitions of the ITEM and GROUP members documenting the columns.

The entries under the heading LENGTH specify the length of a column. The length is the maximum length documented in an ITEM member's definition, or for GROUP members, the total length of the ITEM members it contains. The length of the table or view equals the total length of all the ITEMS and GROUPS. If a member documents an edit, field, or validation procedure there is no entry.

The entries under the heading REMARKS specify the text filed in the DESCRIPTION clause of the members.

**An Example of an SQL/DS Table Layout**

The following is an example of a table layout generated from a SQL-TABLE member:

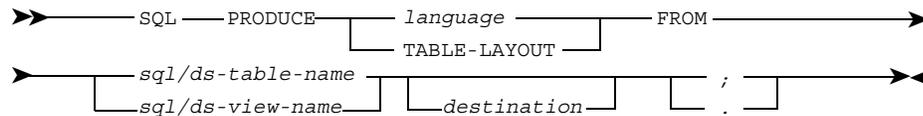
```

* * * * *
*           DESCRIPTION OF TB2-EMPLOYEE           *
* * * * *
PKEY   NULL  NAME                TYPE                LENGTH  REMAR
      N      IT-EMPNO             CHAR (6)         6
      N      IT-FIRSTNME         VARCHAR (10)     13
Y      N      IT-MIDINIT         CHAR (2)         2
      N      IT-LASTNAME         VARCHAR (10)     12          Surname.
      IT-WORKDEPT               CHAR (3)         4
      IT-PHONENO_1              CHAR (4)         5
      IT-JOB                     CHAR (8)         9          Duties.
      IT-SALARY                  DECIMAL (9,2)   6
    
```

**Filing Generated Output in a USER-MEMBER**

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO, PRIVATE, PUBLIC, NEW, APPEND, REPLACE, PRINT, and NOPRINT keywords.

**Syntax of the SQL PRODUCE Command**

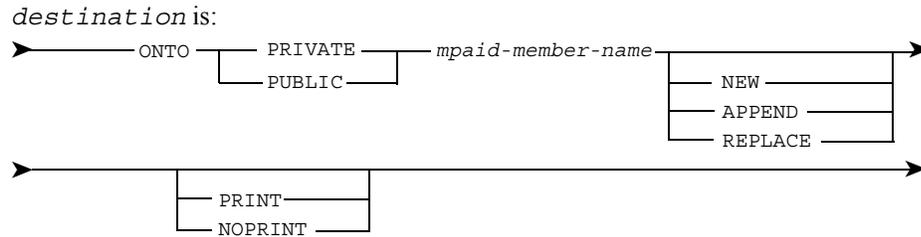


where language is:

- ALC
- ASSEMBLER
- BAL
- COBOL
- PLI
- PL1
- PL/I
- PL/1

sql/ds-table-name is the name of a SQL-TABLE member

sql/ds-view-name is the name of a SQL-VIEW member.



where *mpaid-member-name* is the name of a USER-MEMBER.

## SQL REPORT

Use the SQL REPORT command to produce a SQL Table Report of all or some of the tables in the SQL design.

Refer to "Syntax of the SQL REPORT Command" on page 242 for the syntax of the SQL REPORT command.

### Introduction to the SQL REPORT Command

Use the SQL REPORT command to produce a SQL Table Report of all or some of the tables in the SQL 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 SQL design, the command causes the SQL 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.

### Reporting All the Tables in the Workbench Design Area

To report all the tables in the Workbench Design Area (WBDA), enter:

```
SQL REPORT TABLES ALL ;
```

This outputs the tables in order of WBDA number.

To report all the tables alphanumerically, enter:

```
SQL 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:

```
SQL REPORT TABLES 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 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:

```
SQL REPORT TABLES NAMES name-list ALPHABETICALLY ;
```

For example:

```
SQL REPORT TABLES NAMES DEPARTMENT, OFFICE, EMPLOYEE
ALPHABETICALLY ;
```

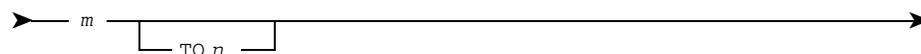
### Reporting Tables Selected by Number

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:

```
SQL 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:

```
SQL 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:

```
SQL 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:

```
SQL REPORT TABLES selection USING FORMAT format-member ;
```

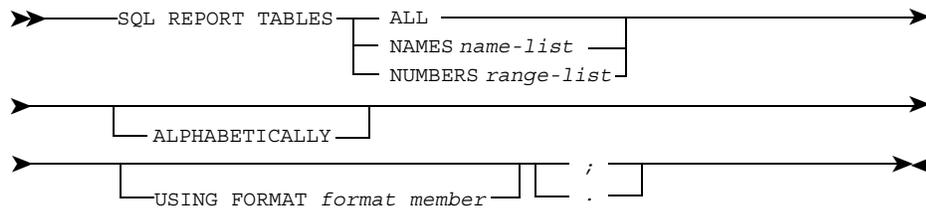
where:

*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 the following:

- ALL
- NAMES *name-list*
- NUMBERS *range-list*.

### Syntax of the SQL REPORT Command



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.

### SQL REVOKE

Refer to "SQL GRANT and SQL REVOKE" on page 190 for details of the SQL REVOKE command.

## SQL SIZE

Use the SQL SIZE command to calculate the total size of a table and the maximum size of each row contained in that table from the definition of a SQL-TABLE member.

### Use

To calculate the total size of a table and the maximum size of each row contained in it, enter:

```
SQL SIZE member-name ;
```

where *member-name* is an encoded SQL-TABLE member.

Row sizes are calculated in bytes and the total size of a table in 4096-byte pages.

The ability to easily calculate row and table sizes makes it much easier to determine the number of pages required for a dbspace, (as defined in the PAGES clause of SQL-DBSPACE members) and plan for future data growth.

The size of a table is estimated by calculating the maximum size of each row in the table and the number of rows the table contains.

The number of rows in the table is taken from the value specified in the CARDINALITY clause of the SQL-TABLE member. If the SQL-TABLE member has no CARDINALITY clause then only the maximum row size is calculated.

The size of a row is determined by the SQL/DS data type of the columns in the table. These are in turn derived from the definition of the ITEMS and GROUPs specified in the COLUMNS clause of the SQL-TABLE member.

The size of a physical row can vary in practice if it contains any variable length columns. The size calculated by the SQL SIZE command uses maximum sizes for all variable columns and so expresses the maximum size for the whole row.

For example, a column with a SQL/DS data type of VARCHAR 6 would be generated from an ITEM with a form-description of CHARACTER 5 TO 6 and no USAGE clause. The SQL SIZE command would calculate the size of the row taking into account the variable length and SQL/DS data type of the column.

If the column can contain null values (that is the keyword NOT-NULL has not been specified in the SQL-TABLE member's definition) it is given an extra byte.

Note that for size calculations a column with a SQL/DS data type of DATE requires four bytes of storage, a column with a SQL/DS data type of TIME three bytes of storage and a column with a SQL/DS data type of TIMESTAMP ten bytes of storage.

The Systems Administrator can tailor the output of the SQL SIZE command so that internal diagnostic output is displayed.

The output of the SQL SIZE command cannot be automatically filed in a USER-MEMBER on the MP-AID.

Refer to "Generating Column Data Types" on page 96 for details of generating column data types.

Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailoring.

### Syntax of the SQL SIZE Command

➤ SQL SIZE *sql/ds-table-name* [ ] ; [ ] ➤

where *sql/ds-table-name* is the name of a SQL-TABLE member.

### SQL SYNONYM

Use the SQL SYNONYM command to generate SQL CREATE SYNONYM or DROP SYNONYM statements from the definition of an SQL-USER member.

#### Use

To generate SQL CREATE SYNONYM statements, enter:

```
SQL CREATE SYNONYM member-name ;
```

To generate SQL DROP SYNONYM statements, enter:

```
SQL DROP SYNONYM member-name ;
```

where *member-name* is an encoded SQL-USER member.

To generate other SQL CREATE or DROP statements use the SQL CREATE or SQL DROP commands.

The generated SQL CREATE SYNONYM and DROP SYNONYM statements can be:

- Printed, or
- Automatically filed in a USER-MEMBER on the MP-AID, or
- Both printed and filed.

To file SQL CREATE SYNONYM or DROP SYNONYM statements in a USER-MEMBER you must specify an ONTO clause in the SQL SYNONYM command.

SQL CREATE SYNONYM and DROP SYNONYM statements can be generated for a synonym on a table or a view. An SQL CREATE SYNONYM or DROP SYNONYM statement is generated for every synonym specified on a SQL-TABLE or SQL-VIEW member in the SYNONYMS clause of the SQL-USER member.

When you have applied an SQL DROP SYNONYM statement to your SQL/DS environment you should remove or update the definition of the relevant SQL-USER member to reflect the changes.

The Systems Administrator can tailor the output of the SQL SYNONYM command so that:

- SQL/DS object names are derived from aliases
- Internal diagnostic output is displayed

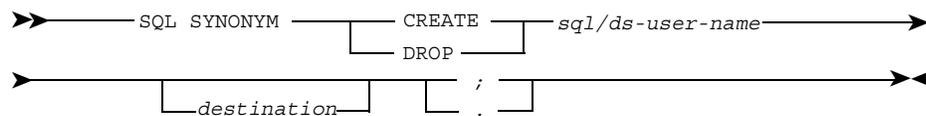
Refer to "SQL-USER" on page 259 for an example of an SQL CREATE SYNONYM statement generated from an SQL-USER member.

Refer to "SQL SYNONYM" on page 244 for details the SYNONYMS clause.

Refer to "Filing Generated Output in a User-member" on page 278 for details of the ONTO clause.

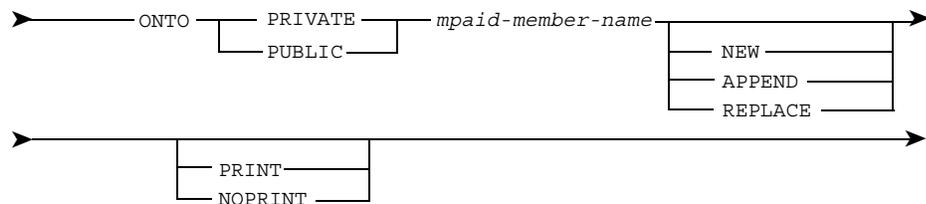
Refer to "Tailoring SQL Statements and SQL/DS Host Language Data Structures" on page 97 for details of tailorability.

### Syntax of the SQL SYNONYM Command



where *sql/ds-user-name* is the name of a SQL-USER member.

*destination* is:



where *mpaid-member-name* is the name of a USER-MEMBER.

### SQL-TABLE

SQL/DS tables are defined in the dictionary as SQL-TABLE members.

Refer to "Syntax of the SQL-TABLE Member Type" on page 257 for the syntax of the SQL-TABLE member type.

### Introduction to SQL-TABLE

To document an SQL/DS table in the dictionary use the SQL-TABLE member type. To define the dictionary member type enter SQL-TABLE at the start of your member definition statement.

Together with SQL-VIEW, it is a member of major interest to the end-user because it contains the user's own data and is not a product of the database administrator or other technical specialist. It is also one of the most used of the SQL/DS member types.

The most important clause available with this member type is the CONTAINS clause. In this clause you specify the ITEM and GROUP members which represent the columns which form the table. In the ITEMS and GROUPS referred to are specified the data types for the columns. The CONTAINS clause, therefore, establishes the relationship between an SQL-TABLE member and ITEM and GROUP members in the dictionary. In a large dictionary, these same GROUPS and ITEMS will typically also form part of other file and database segment definitions. For example, installations with IMS may already have GROUP and ITEM definitions in the dictionary which can now be shared with the SQL/DS environment.

You can also define the owner of a table (using the CREATOR-OWNER clause), and the dbspace in which it is held (using the IN clause). The SQL facilities COMMENT and LABEL are supported by the clauses: SQL-COMMENT and SQL-LABEL.

You will be able to calculate the maximum size of a row in the table and the size of the table (using the SQL SIZE command), if you define a CARDINALITY clause.

You can define and name referential constraints and document the resulting relationships between dictionary members using the CONSTRAINT clause.

Clauses may be declared in any order, except for the following:

- The definition of a referential constraint (in the CONSTRAINT clause) must follow your definition of the columns which make up the table (in the COLUMNS clause)
- An SQL-COMMENT or SQL-LABEL clause applied to a column must follow the definition for that column (either in a group-name EXPAND clause, or in a single-column-clause). By the same token, an SQL-COMMENT or an SQL-LABEL which applies to a table must not immediately follow a column definition, or it will be taken as a COMMENT or a LABEL on that column.

**Note:** \_\_\_\_\_

The name you give to an SQL-TABLE member may be used as the SQL/DS object name.

\_\_\_\_\_

Refer to "Documenting the Columns of SQL/DS Tables and Views" on page 84 for information on documenting the columns of tables.

### **The AS Clause**

Refer to "Defining an AS Clause" on page 277 for details of the AS clause.

### **Defining the Owner of an SQL/DS Table**

Use a CREATOR-OWNER clause to specify the owner of a table. The syntax of the clause is as follows:

➤ CREATOR-OWNER *sql-user* ➤

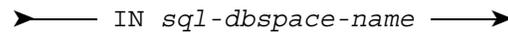
where *sql-user* is the name of a dictionary member of the type SQL-USER, which represents the Authorization ID of the owner of the table. The owner of a table is usually the creator, but in SQL Version 2 Release 2, the owner may not be the creator.

The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of SQL CREATE, ALTER, and DROP statements.

**Defining the Dbspace in which a Table is Located**

Use an IN clause to specify the dbspace in which a table is located. The syntax of the clause is as follows:



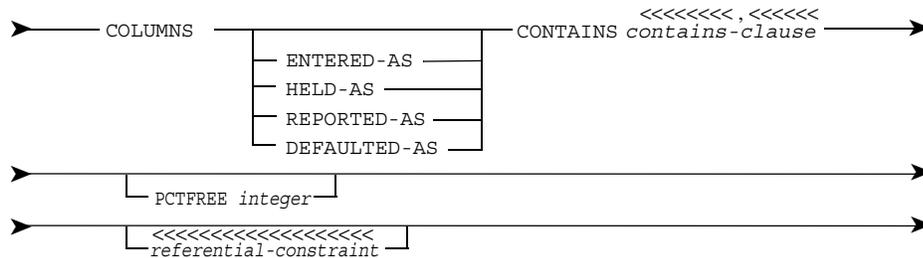
where *sql-dbspace-name* is the name of a dictionary member of the type SQL-DBSPACE, which represents the dbspace in which the table is to be located.

The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

This clause must be present for the successful generation of an SQL CREATE statement.

**Defining the Columns which Make Up an SQL/DS Table**

Use a COLUMNS clause to specify the columns of a table. The syntax of the clause is as follows:



ENTERED-AS, HELD-AS, REPORTED-AS, and DEFAULTTED-AS are the form keywords. One of them may be used to specify the form of all the ITEMS and/or GROUPS which represent the columns of the table. If none is specified, then the DEFAULTTED-AS form of the ITEMS and/or GROUPS is used. If any ITEM or GROUP has no DEFAULTTED-AS form, then the usual ASG-DataManager defaults apply.

For further information, please refer to *ASG-Manager Products Source Language Generation*.

The purpose of the form keyword is to allow the generation of correct data types for host language data structures. Therefore, although this clause is not mandatory for generation, you are strongly advised to include it when you define a table in the dictionary, in order to ensure that the correct data types are generated in SQL PRODUCE statements or host language data structures.

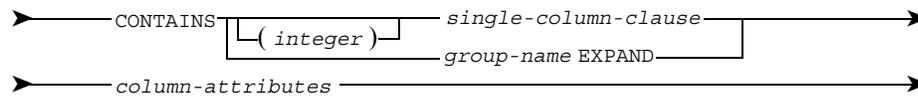
Use the CONTAINS clause to specify the ITEMS or GROUPS which represent the columns of a table.

The PCTFREE parameter is the percentage of space in each index page reserved for later additions or amendments to the primary key. The clause is checked on encoding to ensure that the value specified is within the permitted range. A PCTFREE clause is required only if you define a primary key for a table.

Use the referential-constraint clause to specify any referential constraint(s) which the table may have.

### Defining the ITEMS and/or GROUPs which Represent Columns

Use a CONTAINS clause to specify the ITEMS and/or GROUPs which represent the columns which form the table, and to specify the attributes of these columns. The syntax of the clause is as follows:



where (*integer*) is the number of columns in a 'column set'. The column specified in the following *single-column-clause* will be repeated by the number of times you have specified. On generation of an SQL statement each column produced will be suffixed automatically by a number; the first column will be suffixed by 1, the second by 2, and so on. For example, suppose you have a set of two columns, PERIOD\_1 and PERIOD\_2. If you specify (2) PERIOD in the CONTAINS clause (where PERIOD is the name of the ITEM dictionary member which represents the column) two columns, PERIOD\_1 and PERIOD\_2, will be generated.

Use a *single-column-clause* to specify the name of an ITEM or GROUP dictionary member; each ITEM or GROUP (that is, the ITEMS and/or GROUPs contained by it) represents one column.

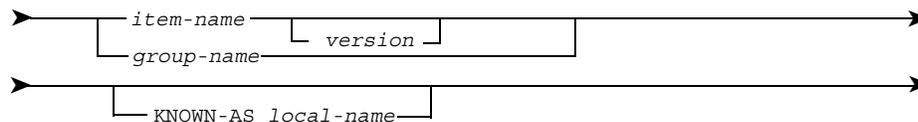
Use the *group-name EXPAND* clause to specify a GROUP member, which represents one or more columns.

A table may have up to 255 generated columns; this figure includes all columns generated as a result of using the *group-name EXPAND* option or the 'column set' option. The clause is checked on generation to ensure that the number of columns specified does not exceed the maximum number permitted.

Use the *column-attributes* clauses to specify attributes which apply to all columns generated from the preceding clause.

### Defining a Single Column in an SQL-TABLE

Use a single-column clause to specify an ITEM or a GROUP dictionary member which represents a single column. The syntax of the clause is as follows:



where:

*item-name* is the name of a dictionary member of the type ITEM

*version* is the version number of the named ITEM. If no version is specified, then version 1 is assumed by default.

*group-name* is the name of a dictionary member of the type GROUP. The GROUP(s) and/or ITEMS(s) contained by this GROUP are combined to represent one column in the table.

The clause is checked on encoding to ensure that the member specified is of one of the correct types (either a GROUP or an ITEM).

Use a KNOWN-AS optional clause to specify a local-name for the ITEM or GROUP. The local-name may be no longer than 18 characters; this is checked on encoding and on generation. This local-name is the name of the column. If you do not specify a local-name in this clause, an alias specified in the ITEM or GROUP will be used as the column-name, (assuming your environment is tailored to generate aliases as external names). If no alias is specified, then the ITEM or GROUP member-name will be used as the column-name (reduced, if necessary, to 18 characters by the name reduction process).

Duplicate column names are not permitted. The clause is checked on generation to ensure that no duplicate column-names are present.

Refer to "Documenting the Columns of SQL/DS Tables and Views" on page 84 for details of how data types are generated from ITEMS and GROUPS.

### Defining Several Columns at Once in an SQL-TABLE

Use a group-name EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

➤ *group-name* EXPAND ➤

where *group-name* is the name of a dictionary member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the table at once; it is a shorthand way of referring to a number of GROUPS and ITEMS (that is, all those contained by the specified GROUP), which will each represent one column in the table. Your installation may already have GROUPS defined in its dictionary which are used in existing applications. These GROUPS may represent records or segments that now need to have counterparts in the SQL environment.

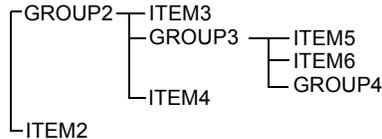
Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

A GROUP to be EXPANDED should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in a table must have a name unique in the table and cannot 'overlay' or share data with any other column in the table. If you do have an ELSE clause, it will be ignored.

A GROUP to be EXPANDED may contain 'nested' GROUPs as well as ITEMs. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS table, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDED group; this is checked on encoding. This means that the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMs and/or GROUPs contained by the GROUP specified, or, failing that, from their dictionary names. The length of any column-name may be no longer than 18 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

Consider the following nested structure which is shown in diagram form for clarity:



Such a structure would be represented in PL/1 as:

```
01 GROUP1,
  02 ITEM1,
  02 GROUP2,
    03 ITEM3
    03 GROUP3
      04 ITEM5
      04 ITEM6
      04 GROUP4
    03 ITEM4
  02 ITEM2
```

In SQL/DS, 'higher' level groups GROUP1, GROUP2, and GROUP3 would be removed to produce a 'flat' one-level structure:

```

01 ITEM1
01 ITEM3
01 ITEM5
01 ITEM6
01 GROUP4
01 ITEM4
01 ITEM2

```

Figure 41 Example of a Nested GROUP Structure

**Note:**

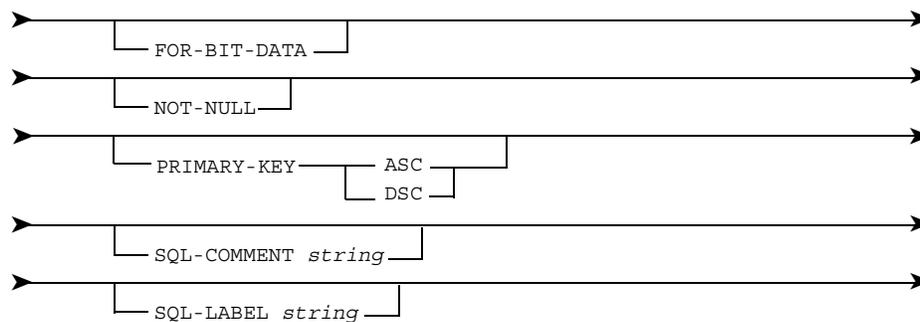
GROUP4, which does not have any lower level, would be considered to be an elementary field; its data type defaults to CHAR(l).

If you consider the original nested structure as a 'tree' and a leaf of the tree as a field which does not have any lower levels, 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.

**Defining SQL/DS Column Attributes**

A number of optional clauses and keywords (the column-attributes clauses) are available to specify the attributes of a column or columns defined in a CONTAINS clause.

The column attributes which you specify will apply equally to all of the columns generated from the preceding CONTAINS clause. Therefore, if you have specified a 'column set' or if you have used the 'group-name EXPAND' option, then the column attributes will apply equally to all of the columns so generated. The syntax of these clauses is as follows:



Use the FOR-BIT-DATA keyword to specify the contents of the column as bit (binary) data for exchange with other systems. You may apply this attribute only to columns specified as having a data-type of CHAR, VARCHAR, or LONG VARCHAR. This requirement is checked on generation of SQL CREATE and ALTER statements.

Use the NOT-NULL keyword to specify that the column may not contain a null value.

Use a PRIMARY-KEY keyword to specify that a column or columns form a primary key. Only one primary-key is permitted in a parent table. The values in the column must be unique and the column must be defined as NOT-NULL.

You can use the SQL-COMMENT and SQL-LABEL clauses in the same way as they are used with regard to a table, except that in this context they apply only to the column(s) defined in the preceding CONTAINS clause.

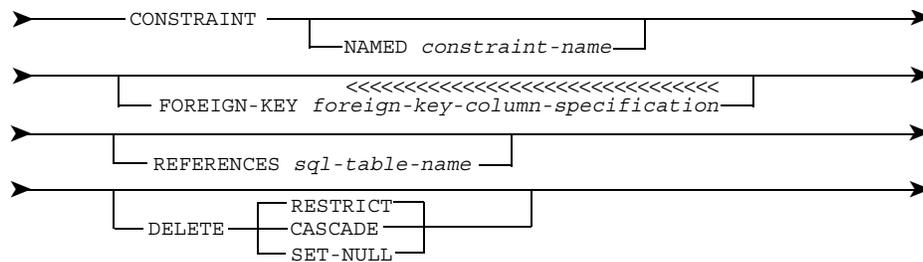
### Defining an SQL/DS Referential Constraint

Use the CONSTRAINT clause to define a referential constraint and foreign keys for a table. You may specify any number of referential constraints for one table. Each referential constraint requires its own CONSTRAINT clause.

Each CONSTRAINT clause allows you to specify, optionally, a constraint-name, one or more columns to form the foreign key, the table being referenced and the associated DELETE rule.

You can name a referential constraint without specifying the column or columns that form the foreign key. Thus you can set up SQL-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.

The syntax of the clause is as follows:



where:

*constraint-name* is the name by which the referential constraint will be known to SQL/DS. The constraint-name may be no longer than 8 characters; this is checked on encoding. Each constraint-name must be unique within a table. If you omit to specify a constraint-name, a default name will be generated by SQL/DS.

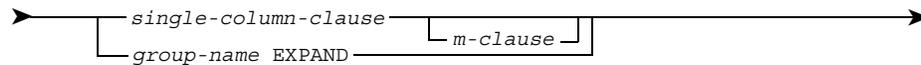
*foreign-key-column-specification* is the specification of each column which makes up the foreign key, and consists of the same information as is used to define the column in the single-column clause or the group-name EXPAND clause of the COLUMNS clause.

*sql-table-name* is the name of a dictionary member of the type SQL-TABLE, which represents the parent table to which the foreign key refers. One REFERENCES clause must be present for each foreign key specified. The SQL-TABLE you name must exist, and must include a valid CREATOR-OWNER clause, for successful generation of an SQL statement.

Use a DELETE clause to specify the DELETE rule to be established by SQL/DS for the referential constraint. The keywords are the same as those used by SQL/DS and they have the same meanings. If you omit to specify a DELETE option, none will be generated, and the SQL/DS default will apply.

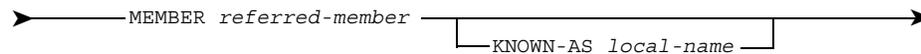
**Defining an SQL/DS Foreign Key**

Use a FOREIGN-KEY clause in a dependent table to define the column(s) which form the foreign key. When you generate an SQL statement from a table with foreign keys, the foreign key specification must already be present in an identical form in the single-column clause or the group-name EXPAND clause (except that the 'version' specification is omitted in the foreign key specification). The syntax of the clause is as follows:



where *single-column-clause* and *group-name EXPAND* are used in the same way as they are used in the COLUMNS clause to define the columns.

Use an *m-clause* to establish a correspondence between the column(s) forming the primary key of the parent table defined in the REFERENCES clause (that is, the column to which the foreign key refers) and the column(s) of the foreign key being specified. The syntax is as follows:



where:

*referred-member* is the name of an ITEM or GROUP dictionary member, which represents the column which forms the primary key in the parent table

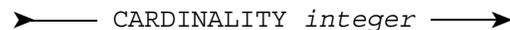
*local-name* is the local name for that column (that is, the column-name). The clause is checked on encoding to ensure that the referred-member specified is of the correct type (that is, an ITEM or a GROUP). The length of the local-name specified in the KNOWN-AS clause may be no longer than 18 characters; this is checked on encoding.

The m-clause can be applied only to a single-column clause or to a column set.

You are particularly advised to use an m-clause in cases where the names of the foreign key columns differ from the names of the corresponding primary key columns in the referenced tables, or where different ITEM and/or GROUP members represent the foreign key columns of one table and the corresponding primary key columns of the referenced table. In these cases, if an m-clause is included, the correspondence between the members is clearly established in the dictionary.

**Specifying an Estimate of the Number of Rows in an SQL/DS Table**

Use the CARDINALITY clause to specify an estimate of the number of rows which will be contained in the table. The syntax of the clause is as follows:



where *integer* is the number of rows which you foresee the table will contain. The integer may be no greater than 2147483647.

This clause must be present if you wish to estimate the total size of the table using the SQL SIZE command.

### Defining an SQL-COMMENT on a Table or a Column

Use an SQL-COMMENT clause to specify that a table or column is to have an associated SQL comment. The syntax of the clause is as follows:

► SQL-COMMENT '*string*' ►

where '*string*' is the comment. The comment must be a character string consisting of no more than 254 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use an SQL-COMMENT clause to qualify a column, it must follow the contains-clause which defines the column.

This clause must be present if you want to generate SQL COMMENT ON statements from an SQL-TABLE member definition.

### Defining an SQL-LABEL on a Table or a Column

Use an SQL-LABEL clause to specify that a table or column is to have an associated SQL LABEL. The syntax of the clause is as follows:

► SQL-LABEL '*string*' ►

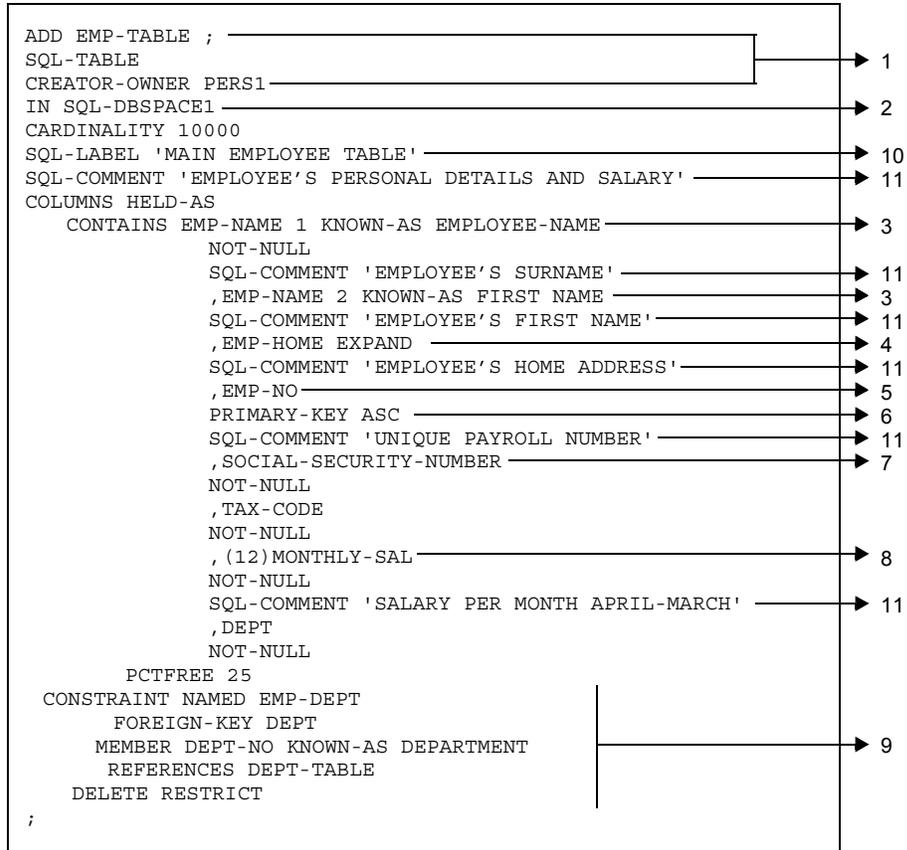
where '*string*' is the label. The label must be a character string consisting of no more than 30 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-LABEL clause to qualify a column, it must follow the contains-clause which defines the column.

This clause must be present if you want to generate SQL LABEL ON statements from an SQL-TABLE member definition.

Example: SQL-TABLE Definition and SQL Generation

Dictionary Definition:



Generated SQL Syntax:

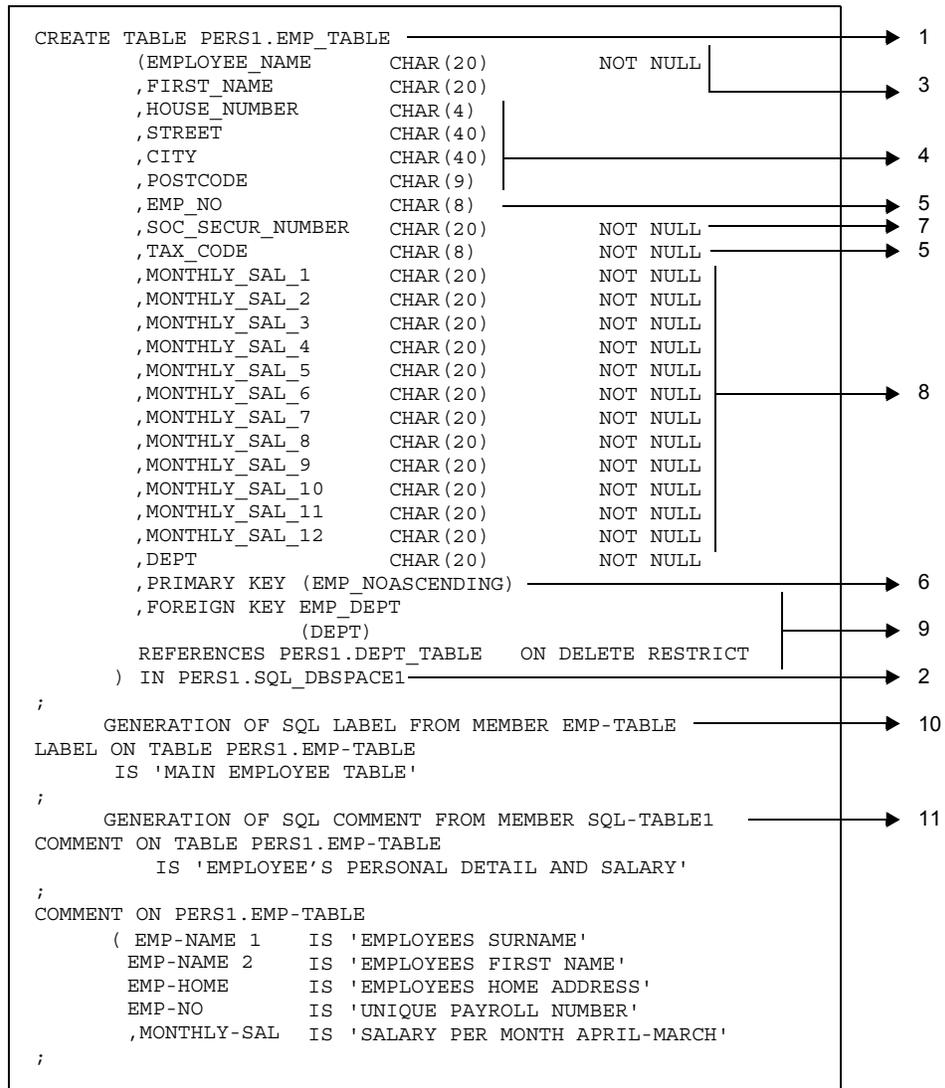


Figure 42 SQL-TABLE Definition and SQL Generation

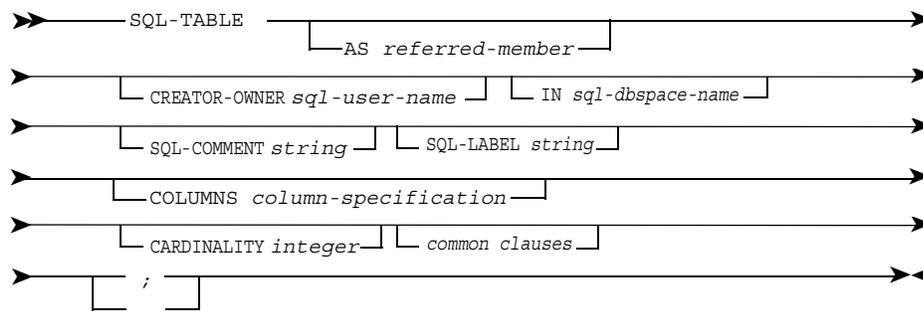
1. The derived name for the table is the SQL-TABLE member name qualified by the name of the SQL-USER member which represents the owner's SQL/DS Authorization ID.
2. The name of the dbspace in which the table is located is taken directly from the table definition and qualified by the Authorization ID of the owner of the dbspace.
3. The column-names are taken from the KNOWN-AS names which qualify each of the specifications of the versions of the ITEM which represent the columns.
4. The four columns `HOUSE_NUMBER`, `STREET`, `CITY`, and `POSTCODE` are each represented by an ITEM member contained by the GROUP `EMP-HOME`.

5. The column-names EMP\_NO and TAX\_CODE are taken directly from the dictionary definition.
6. The PRIMARY-KEY specification is taken directly from the specification in the member definition.
7. The column-name SOC\_SECUR\_NUMBER is the reduced form of the dictionary name of the ITEM which represents it (that is, the name is taken directly from the dictionary definition, but has been subject to the Name Reduction Process).
8. The 12 occurrences of MONTHLY-SAL specified in the dictionary definition result in the generation of 12 columns, each called MONTHLY\_SAL; each column is uniquely identified by the integer suffixed to it.
9. Details of the foreign key are taken directly from the CONSTRAINT clause of the member definition.
10. The SQL-LABEL associated with the table is taken directly from the specification in the member definition.
11. The SQL-COMMENTS associated with the table and columns are taken directly from the specifications in the member definition.

**Note:**

Automatic output of LABELs and COMMENTs from an SQL CREATE command is dependent upon the Systems Administrator having set up the command variables CM\_LABELOPT and CM\_COMMENTOPT in the tailorability executives appropriately.

**Syntax of the SQL-TABLE Member Type**



where:

*referred-member* is the name of an SQL-TABLE dictionary member

*sql-user-name* is the name of an SQL-USER dictionary member

*sql-dbspace-name* is the name of an SQL-DBSPACE dictionary member

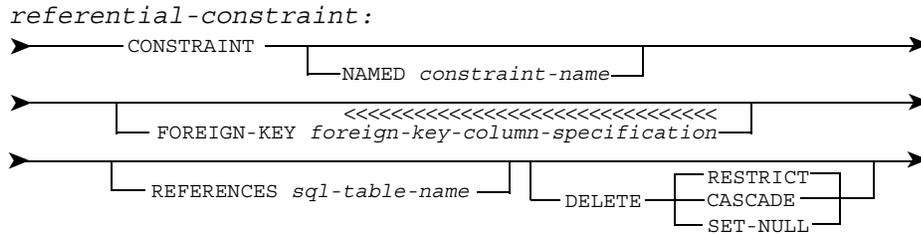
*string* (in the SQL-COMMENT clause) is a character string of no more than 254 characters



where:

*string* (in the SQL-COMMENT clause) is a character string of no more than 254 characters

*string* (in the SQL-LABEL clause) is a character string of no more than 30 characters.

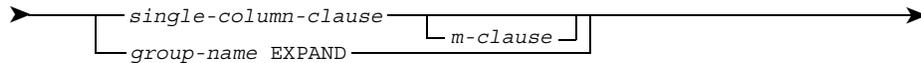


where:

*constraint-name* is the name of the constraint, consisting of no more than 18 characters

*sql-table-name* is the name of an SQL-TABLE dictionary member.

*foreign-key-column-specification*:

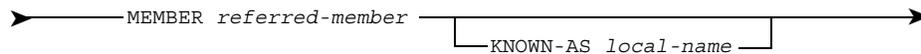


where:

*single-column clause* is as above

*group-name* is the name of a GROUP dictionary member.

*m-clause*:



where:

*referred-member* is the name of an ITEM or a GROUP dictionary member

*local-name* consists of no more than 18 characters.

## SQL-USER

SQL/DS users are defined in the dictionary as SQL-USER members.

Refer to "Syntax of the SQL-USER Member Type" on page 261 for the syntax of the SQL-USER member type.



Example: SQL-USER Definition and SQL

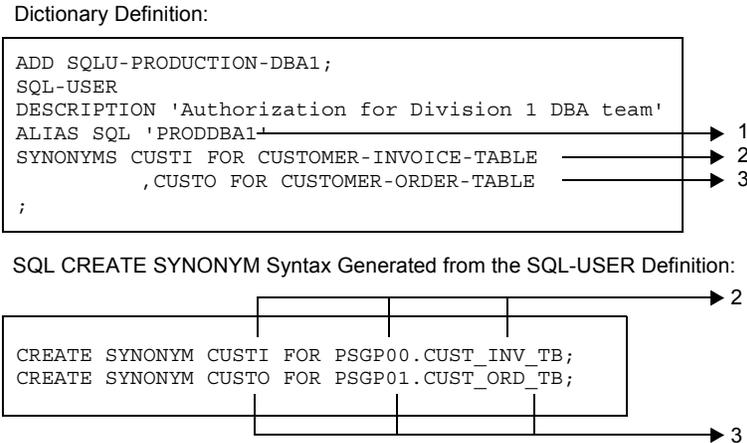


Figure 43 SQL-USER Definition and SQL CREATE SYNONYM Generation

1. The user’s SQL ALIAS, 'PRODDBA1', is not used in the generation of this SQL statement. Instead, it serves as a record in the dictionary of the actual Authorization ID.

**Note:**

SYNONYM CREATE statements will have to be executed in SQL/DS by the Authorization ID 'PRODDBA1' to reflect the creator of the synonyms.

2. The first SYNONYM command is generated with CUSTI as the synonym name, taken exactly as entered in the dictionary definition shown in this example. The fully-qualified table name is derived by taking the SQL ALIAS (PSGP00) of the SQL-USER member referred to in the CREATOR-OWNER clause of the SQL-TABLE CUSTOMER-INVOICE-TABLE. Neither of these two members are shown here.

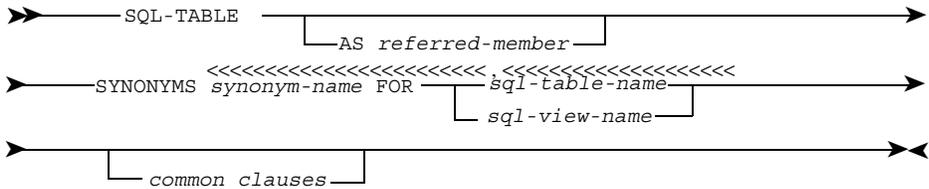
**Note:**

To generate an SQL CREATE SYNONYM statement, enter:

```
SQL SYNONYM CREATE SQLU-PRODUCTION-DBA1;
```

3. The second SYNONYM command is generated in the same way as described in 2 above.

Syntax of the SQL-USER Member Type



where:

*referred-member* is the name of an SQL-USER dictionary member

*synonym-name* is a synonym, consisting of no more than 18 characters

*sql-table-name* is the name of an SQL-TABLE dictionary member

*sql-view-name* is the name of an SQL-VIEW dictionary member.

## **SQL-VIEW**

SQL/DS views are defined in the dictionary as SQL-VIEW members.

Refer to "Syntax of the SQL-VIEW Member Type" on page 275 for the syntax of the SQL-VIEW member type.

### **Introduction to SQL-VIEW**

To document an SQL/DS view in the dictionary use the SQL-VIEW member type. To define the dictionary member type enter SQL-VIEW at the start of your member definition statement.

SQL/DS views are possibly the SQL/DS objects most often used by end users, therefore the SQL-VIEW member type is, with SQL-TABLE, of major importance.

The SQL-VIEW member type bears many similarities to SQL-TABLE.

The main similarity is that the columns which form the view are defined in a column-specification clause, in which you specify the ITEM and GROUP members which represent the columns which form the view. As with SQL-TABLE, therefore, this clause establishes relationships between an SQL-VIEW member and ITEM and GROUP members in the dictionary. Clauses which are particular to the SQL-VIEW member-type (the SELECT, FROM, WHERE, and HAVING clauses, which define a view subselect) also establish relationships between a view and the base table(s) or view(s) which the columns, that form the view, belong to.

You can define the owner of a view, by using the CREATOR-OWNER clause. The SQL/DS functions COMMENT and LABEL are supported in an SQL-VIEW member, as in an SQL-TABLE member, by the SQL-COMMENT, and SQL-LABEL clauses.

The SQL subselect part of the SQL statement which is generated from an SQL CREATE VIEW command is supported by the optional SELECT clause and its dependent FROM, WHERE, and HAVING clauses.

You may declare the clauses in any order, except that the column-attributes which qualify a particular column must be declared immediately following the definition of that column. This is especially important if you use SQL-COMMENT and SQL-LABEL clauses to qualify columns. If they do not appear immediately after the column-specification of the column to which they apply, they will be taken as applying to the whole view.

**Note:** \_\_\_\_\_

The name you give to an SQL-VIEW member may be used as the SQL/DS object name.  
\_\_\_\_\_





*version* is the version number of the named ITEM. The version you specify must correspond with a version specified in the ITEM. If no version is specified, then version 1 is assumed by default.

*group-name* is the name of a dictionary member of the type GROUP. The GROUPs and/or ITEMs contained by this GROUP are combined to represent one column in the view.

The clause is checked on encoding to ensure that the member specified is of one of the correct types (either a GROUP or an ITEM).

Use a KNOWN-AS optional clause to specify a local-name for the ITEM or GROUP. The local-name may be no longer than 18 characters; this is checked on encoding and on generation. This local-name is the name of the column, unless a column-name is specified in the COLUMN-NAME clause.

If no column-name is specified at all in the dictionary definition (either in the COLUMN-NAME clause, or in the KNOWN-AS clause), then an alias specified in the ITEM or GROUP will be used as the column-name (assuming your environment is tailored to generate aliases as external names). If no alias is specified the ITEM or GROUP member-name will be used as the column-name (reduced, if necessary, to 18 characters by the Name Reduction Process).

Duplicate column names are not permitted. The clause is checked on generation to ensure that no duplicate column-names are present.

Refer to "Documenting the Columns of SQL/DS Tables and Views" on page 84 for details of how ITEMs and GROUPs are used in SQL/DS views.

### Defining Several Columns at Once in an SQL/DS View

Use a *group-name* EXPAND clause to specify a number of columns at once. The syntax of the clause is as follows:

► *group-name* EXPAND ►

where *group-name* is the name of a dictionary member of the type GROUP. The purpose of this clause is to facilitate the generation of several columns in the view at once; it is a shorthand way of referring to a number of GROUPs and ITEMs (that is, all those contained by the specified GROUP), which will each represent one column in the view. Your installation may already have GROUPs defined in its dictionary which are used in existing applications. These GROUPs may represent records or segments that now need to have counterparts in the SQL/DS environment.

Because of the simpler 'flat-file' structures supported by SQL/DS you need to observe the following points.

A GROUP to be EXPANDED should not contain any ELSE clauses. These give rise to record 'overlays', that is, records in which certain fields may share the same areas of physical storage. In SQL/DS such a concept has no meaning, since a column in a view must have a name unique in the view and cannot 'overlay' or share data with any other column in the view. If you do have an ELSE clause, it will be ignored.

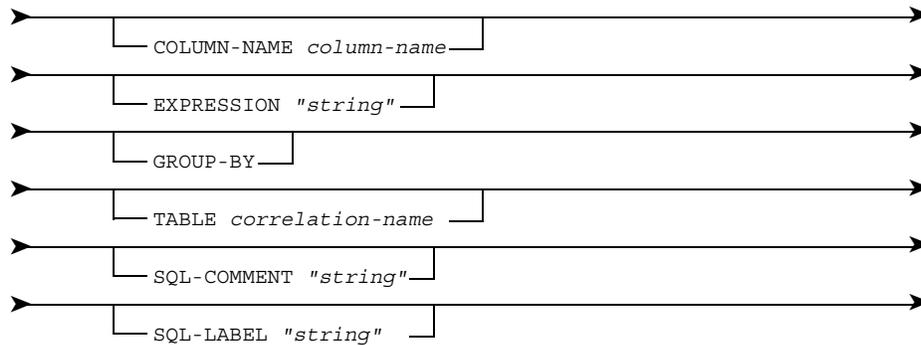
A GROUP to be EXPANDED may contain 'nested' groups as well as items. Nesting can continue to any depth; the only limit is the amount of memory available. However, whereas in segments and host language data structures, such nesting is meaningful, in an SQL/DS view, it is not. Therefore, intermediate levels in the data structure are removed, in order to generate a 'flat' structure.

You may not specify a KNOWN-AS clause or a version for an EXPANDED group; this is checked on encoding. Therefore, the column-names generated from a group-name EXPAND clause are taken either from the KNOWN-AS names of the ITEMS and/or GROUPs contained by the GROUP specified, or, failing that, from their dictionary names. The length of any column-name may be no longer than 18 characters; this is checked on encoding. The clause is also checked on generation to ensure that the length of the derived names is compatible with the external environment.

### Defining Column Attributes for an SQL/DS View

A number of optional clauses (the column-attributes clauses) are available to specify the attributes of a column or columns defined in a CONTAINS clause.

The Column attributes which you specify will apply equally to all of the columns generated from the preceding CONTAINS clause. Therefore, if you have specified a 'column set' or if you have used the 'group-name EXPAND' option, then the column attributes will apply equally to all of the columns so generated. The syntax of these clauses is as follows:



where *column-name* is the name by which the column will be known. The column-name may be no longer than 18 characters; this is checked on encoding. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

If you omit to specify a column-name, then the column-name generated is taken from the KNOWN-AS name, an alias, or the dictionary member name, of the ITEM or GROUP which represents the column, according to the rules for the derivation of external names.

**Note:** \_\_\_\_\_

Since each column-name must be unique to a view, you may not use this clause after the 'group-name EXPAND' option.

\_\_\_\_\_

where "string" in the EXPRESSION clause is a valid SQL expression enclosed in quotes. The string can be a maximum of 255 characters long. This string constitutes the expression which is used to calculate the values to be contained in the column (the values being the result of the calculation performed on one or more of the other columns in the view). Where a view contains such a column you should have a corresponding entry in the CONTAINS clause defining an ITEM, from which the correct data type can be derived on generation of an SQL PRODUCE command.

The keyword GROUP-BY is used to indicate that the column is to be a 'GROUP BY' column: this keyword has the same meaning as for SQL/DS.

where *correlation-name* is the correlation-name of the table or view from which a column is being selected. The correlation-name may be no longer than 18 characters; this is checked on encoding. The purpose of this clause is to resolve any ambiguity which may arise when, for instance, two columns with the same name are present in two tables or views specified in the FROM clause. The TABLE clause, therefore, corresponds with the CORRELATION-NAME clause of the FROM clause: the correlation-name given to a table or view in that clause must be the same as the one given to the table or view in this clause.

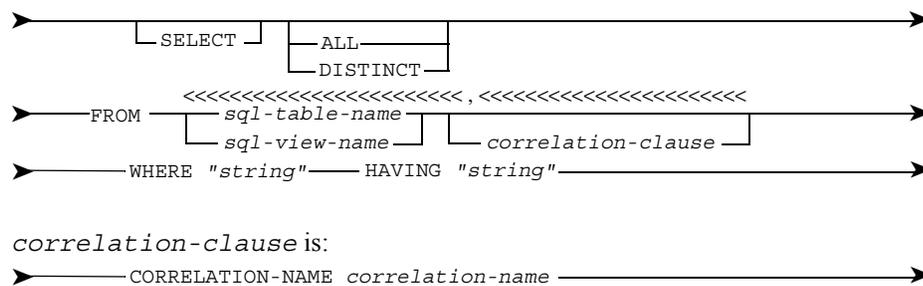
You can use the SQL-COMMENT and SQL-LABEL clauses in the same way as they are used with regard to a view, except that in this context they apply only to the column(s) defined in the preceding CONTAINS clause.

Refer to "SQL COMMENT and SQL LABEL" on page 180 for details of how to define an SQL-COMMENT or SQL-LABEL.

Refer to "Naming Conventions for SQL/DS Members" on page 89 for details of naming conventions and the derivation of external names.

### Defining an SQL/DS View Subselect

You can define the subselect of a view in the dictionary by using the SELECT, FROM, CORRELATION-NAME, WHERE, and HAVING clauses. The syntax of the "subselect" clauses is as follows:



In the SELECT clause, the keywords ALL and DISTINCT have the same meanings as in SQL/DS. The clause is checked on encoding to ensure the option specified is a valid one. The keyword SELECT should not be included in the member definition, if neither option is required, if you omit the clause completely, the keywords AS SELECT will be generated with neither ALL nor DISTINCT.

In the FROM clause, *sql-table-name* is the name of a dictionary member of the type SQL-TABLE, and *sql-view-name* is the name of a dictionary member of the type SQL-VIEW. The clause is checked on encoding to ensure that the member specified is of the correct type. The clause is checked on generation to ensure that the length of the derived name is compatible with SQL/DS requirements.

These are the tables and/or views upon which a view is based. The SELECT clause must be present for the successful generation of an SQL CREATE statement, and the tables and views specified in it must exist, and have valid CREATOR-OWNER clauses.

In the CORRELATION-NAME clause, *correlation-name* is the correlation-name to be assigned to the sql-table or sql-view specified in the FROM clause. The correlation-name may be no longer than 18 characters; this is checked on encoding. The correlation-name which you define here must correspond with that defined in the TABLE clause. The correlation-name is generated only if the TABLE clause is present.

In the WHERE and HAVING clauses, "*string*" consists of a valid SQL expression, enclosed in quotes. This string constitutes the 'where-clause' or the 'having-clause' of the subselect in an SQL CREATE VIEW statement. It should contain the valid SQL expression of a 'where' search condition or a 'having' search condition, as appropriate.

**Note:** \_\_\_\_\_

The validity of the SQL expression is not checked, when an SQL-VIEW member is encoded, or when an SQL statement is generated.

---

### **Defining an SQL-COMMENT on a View or a Column**

Use an SQL-COMMENT clause to specify that a view or a column is to have an associated SQL comment. The syntax of the clause is as follows:

➤ SQL-COMMENT "*string*" ➤

where "*string*" is the comment. The comment must be a character, string consisting of no more than 254 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-COMMENT clause to qualify a column, it must follow the contains-clause which defines the column. To qualify a view, put the clause at the start of the data definition statement, before the column-specification clauses.

This clause must be present if you want to generate SQL COMMENT ON statements from an SQL-VIEW definition.

### **Defining an SQL-LABEL on a View or a Column**

Use an SQL-LABEL clause to specify that a view or a column is to have an associated SQL-LABEL. The syntax of the clause is as follows:

➤ SQL-LABEL "*string*" ➤

where "*string*" is the label. The label must be a character string consisting of no more than 30 characters, enclosed in quotes. The length of the string is checked on encoding.

If you use the SQL-LABEL clause to qualify a column, it must follow the CONTAINS clause which defines the column. To qualify a view, put the clause at the start of the data definition statement, before the column-specification clauses.

This clause must be present if you want to generate SQL LABEL ON statements from an SQL-VIEW definition.

### *Examples of SQL-VIEW Definitions and SQL Generation*

#### **Example 1: SQL-VIEW Definition Containing a Join and SQL Generation**

In this example, a view is being defined which will have columns giving employees' names, their payroll numbers, their departments, and their annual salaries. The view must be defined as the join of two tables. The two tables are:

EMP-TABLE (correlation name E) which contains the following columns:

EMP-NAME  
SOC- SEC-NO  
DEPT-NO  
MONTHLY-SAL

and DEPT-TABLE (correlation-name D) which contains the following columns:

DEPT-NO  
DEPT-NAME.

The view is to have four columns, as follows:

NAME (derived from EMP-NAME in EMP-TABLE)

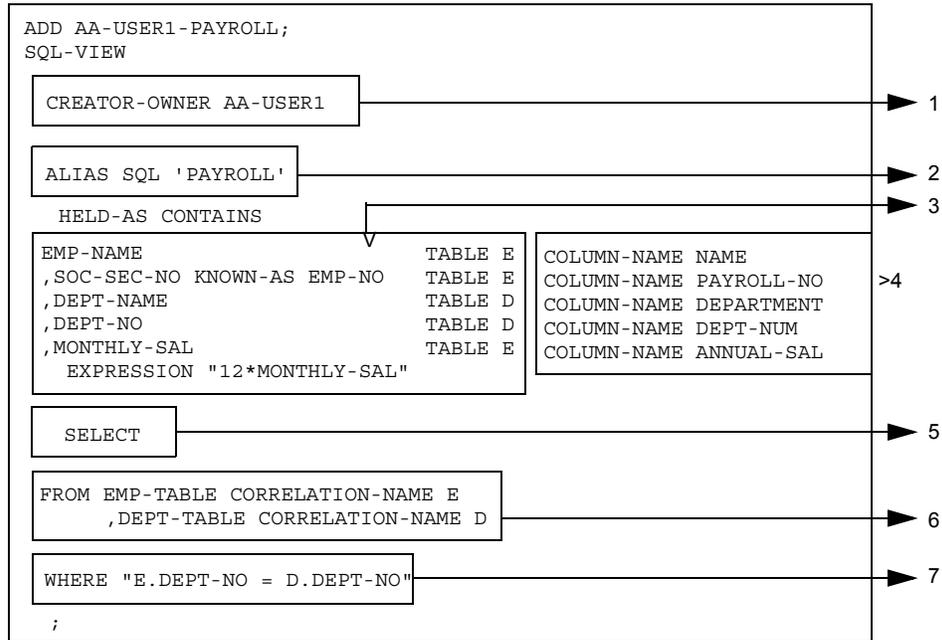
PAYROLL-NO (derived from SOC-SEC-NO in EMP-TABLE)

DEPARTMENT (derived from DEPT-NAME in DEPT-TABLE)

ANNUAL-SAL (derived by multiplying the column MONTHLY-SAL in EMP-TABLE by 12).

The join column is DEPT-NO, which is present in both tables. That is, to obtain the department name the employee table must be joined with the department table by matching the department numbers.

Dictionary Definition:



Generated SQL Syntax:

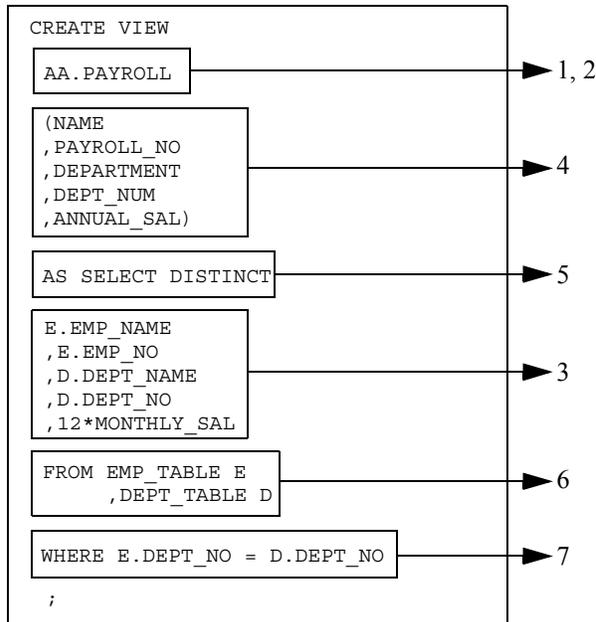


Figure 44 SQL-VIEW Containing a Join and SQL Generation

- 1, 2. The derived name for the view is the SQL ALIAS of the SQL-VIEW member, qualified by the SQL ALIAS defined in the SQL-USER member which represents the owner's SQL/DS Authorization ID.

3. The columns to be selected are qualified by the correlation names of the tables to which they belong.
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 SELECT option is taken directly from the member definition.
6. The tables from which the columns are selected and their correlation-names are taken directly from the FROM clause in the member definition.
7. The WHERE SQL expression is taken directly from the WHERE clause.

**Example 2: SQL-VIEW Definition Containing a GROUP BY and SQL Generation**

In this example, a view is being defined which will have columns giving 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 following columns:

DEPT-NO  
DEPT-NAME  
EMP-NO.

The view's three columns are:

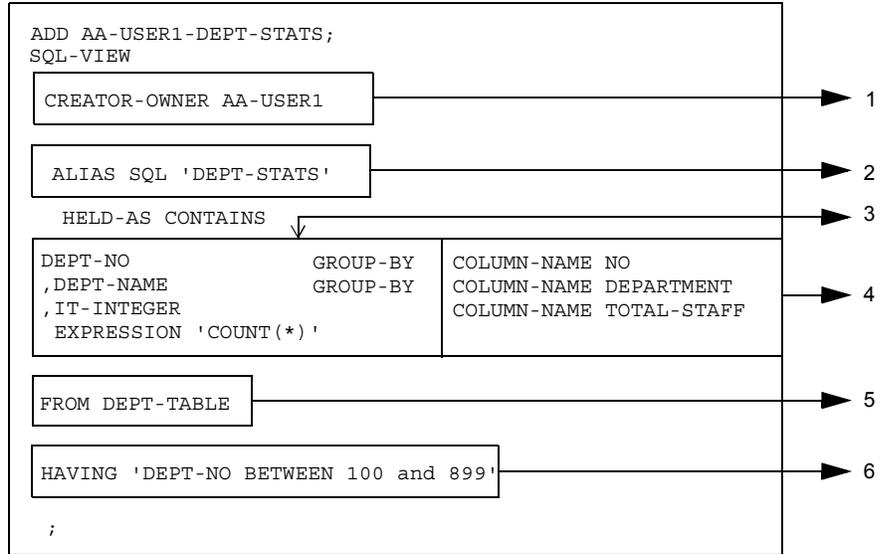
NO (derived from DEPT-NO)

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 to the total number of rows for each department.

Dictionary Definition:



Generated SQL Syntax:

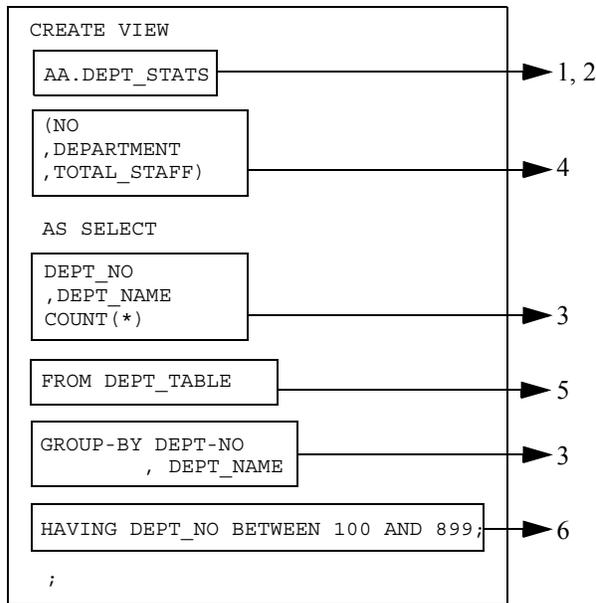


Figure 45 SQL-VIEW Containing a GROUP BY and SQL Generation

- 1, 2. The derived name for the view is the SQL ALIAS of the SQL-VIEW member, qualified by the SQL ALIAS defined in the SQL-USER member which represents the owner's SQL/DS Authorization ID.

3. The columns to be selected are taken directly from the CONTAINS clause. The third column, which will contain 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 data definition.
5. The table from which the columns are drawn is taken directly from the FROM clause in the member definition.
6. The HAVING SQL expression is taken directly from the HAVING clause.

**Example 3: SQL-VIEW Definition Containing a GROUP BY and a Join and SQL Generation**

In this example, a view is being defined which will have columns giving 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 as the join of two tables:

DEPT-TABLE (correlation-name D) which contains the following columns:

DEPT-NO  
DEPT-NAME  
PROJ-NO

and PROJ-TABLE (correlation-name P) which contains the following columns:

PROJ-NO  
PROJ-NAME  
TOTAL-HOURS.

The view's three columns are:

DEPT-NO (derived from DEPT-NO in DEPT-TABLE)

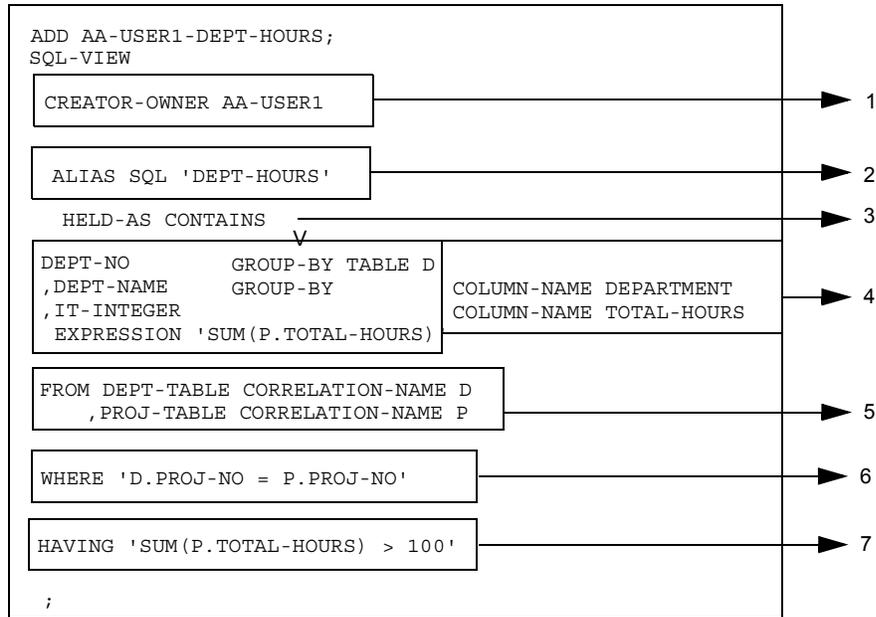
DEPARTMENT (derived from DEPT-NAME in DEPT-TABLE)

TOTAL-HOURS (derived by summing the TOTAL-HOURS column in the table obtained by joining DEPT-TABLE and PROJ-TABLE on the common column PROJ-NO).

The join column is PROJ-NO, which is present in both tables. The view's subselect must also have a GROUP-BY clause to be able to sum the total hours in one department and also a HAVING clause to be able to exclude all the groups which do not have total hours exceeding 100 hours.

Notice the use of the ITEM member IT-INTEGER. Its purpose is merely to act as a 'place marker' in the CONTAINS clause and to ensure that the correct data type is generated for host structures.

Dictionary Definition:



Generated SQL Syntax:

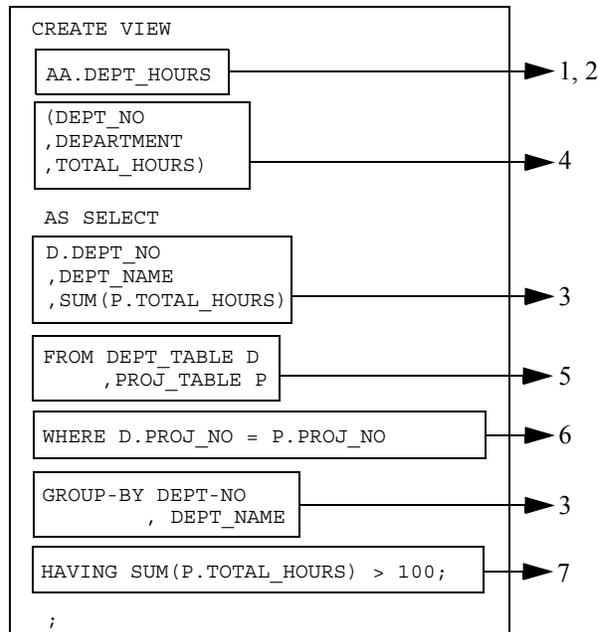
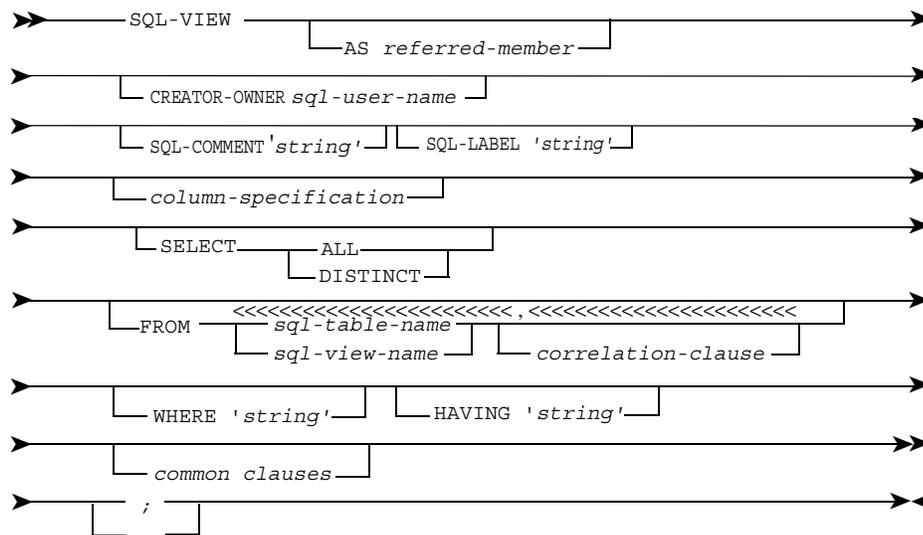


Figure 46 SQL-VIEW Containing a Join and a GROUP BY and SQL Generation

- 1,2. The derived name for the view is the SQL ALIAS of the SQL-VIEW member, qualified by the SQL ALIAS defined in the SQL-USER member which represents the owner's SQL/DS authorization ID.

3. The columns to be selected are taken directly from the CONTAINS clause. The third column, which will contain 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 clauses in the data definition, or, in the case of the first column, directly from the dictionary 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 SQL expression is taken directly from the WHERE clause.
7. The HAVING SQL expression is taken directly from the HAVING clause.

**Syntax of the SQL-VIEW Member Type**



where:

*referred-member* is the name of an SQL-VIEW dictionary member

*sql-user-name* is the name of an SQL-USER dictionary member

*sql-table-name* is the name of an SQL-TABLE dictionary member

*sql-view-name* is the name of an SQL-VIEW dictionary member

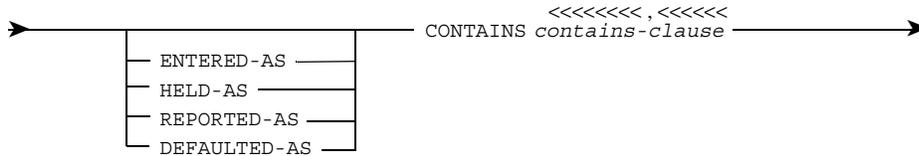
'string' (in the WHERE clause) is a character string

'string' (in the HAVING clause) is a character string

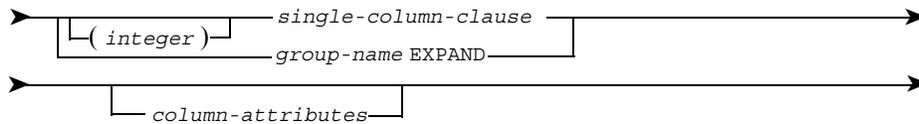
'string' (in the SQL-COMMENT clause) is a character string of no more than 254 characters

'string' (in the SQL-LABEL clause) is a character string of no more than 30 characters.

*column-specification:*



*contains-clause:*

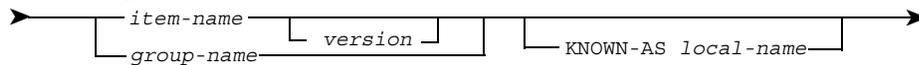


where:

*integer* is the number of columns in a 'column set'

*group-name* is the name of a GROUP dictionary member.

*single-column clause:*



where:

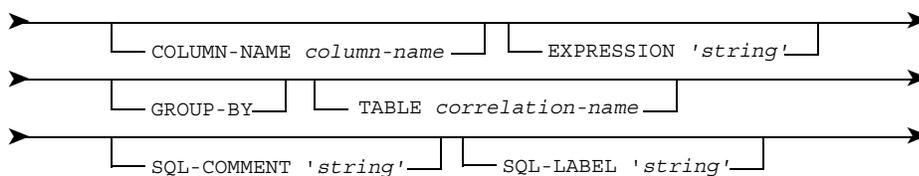
*item-name* is the name of an ITEM dictionary member

*group-name* is the name of a GROUP dictionary member

*version* is an integer in the range 1 to 15

*local-name* consists of no more than 18 characters.

*column-attributes:*



where:

*column-name* is the name of the column, consisting of no more than 18 characters

'string' (in the EXPRESSION clause) is a character string of no more than 255 characters

*correlation-name* is the name of a correlated table or view, consisting of no more than 18 characters

'*string*' (in the SQL-COMMENT clause) is a character string of no more than 254 characters

'*string*' (in the SQL-LABEL clause) is a character string of no more than 30 characters.

*correlation-clause*:

➤ — CORRELATION-NAME *correlation-name* — ➤

where:

*correlation-name* is the name of a correlation name, consisting of no more than 18 characters.

## Defining an AS Clause

Use the AS clause when you are defining a dictionary member which contains one or more clauses which are duplicated in another member of the same type.

When an SQL statement is generated from a member containing an AS clause, information is extracted from an already defined member (the S member). This avoids re-keying of information, and saves space in the dictionary.

To define a dictionary member using the AS clause:

- Declare the member type of the member you want to create
- Declare the clauses which are unique to this member, or whose values differ from the AS member's
- Use the AS clause to refer to the member from which you wish to extract the remaining clauses.

The syntax of the clause is as follows:

➤ — AS *referred-member* — ➤

where *referred-member* is the name of a dictionary member of the same type.

For example, suppose you have defined a table in a member named PROJECT, and you want to define another table called SPECIAL-PROJECT which is identical in every respect to PROJECT. When you define SPECIAL-PROJECT, you can use the AS clause to refer to PROJECT. On generation of a CREATE statement, the clauses in PROJECT are used to create SPECIAL-PROJECT.

The two tables PROJECT and SPECIAL-PROJECT would be identical.

## **Filing Generated Output in a User-member**

You can automatically file the output you have generated in a private or public USER-MEMBER on the MP-AID by specifying an ONTO clause. You can also stop the output you have generated being printed.

The generated output can be filed in either a private or a public USER-MEMBER. Specify the keyword PUBLIC to file the output in a public USER-MEMBER or PRIVATE to file it in a private USER-MEMBER.

The USER-MEMBER can be either a new or an existing member. Users with different Logon Identifiers can create distinct USER-MEMBERS with the same name.

To file the generated output in a new USER-MEMBER either specify the keyword NEW (for readability) or do not specify any keyword.

To file the generated output in an existing USER-MEMBER specify either of the keywords APPEND or REPLACE. The output generated will replace the current contents of the member if REPLACE is specified or be appended to it if APPEND is specified. A new USER-MEMBER will be created if the member does not already exist.

When appending or replacing the contents of an existing USER-MEMBER the user who created that member can change it from a private to a public USER-MEMBER, or the reverse, by specifying either PRIVATE or PUBLIC.

To print the generated output either specify the keyword PRINT (for readability) or do not specify any keyword. To stop the generated output being printed specify NOPRINT.

Manager Products messages are not filed in the USER-MEMBER and are always printed.

---

## Appendix

---

# The Manager Products Name Reduction Process

## Introduction to the Name Reduction Process

The Name Reduction Process is invoked automatically to ensure that the length of a name which will be used in an external environment is no longer than the maximum acceptable to the relevant environment. Name reduction takes place when you:

- Generate Database Definition Language (DDL) statements from dictionary members
- Generate host language data structures in COBOL, Assembler, or PL/I from dictionary members
- Populate the dictionary with members generated from the WBDA.

The principle of the Name Reduction Process is, wherever possible, to recognize constituent parts of names and to shorten each part. As a result, duplicate names are less likely to arise than if a simple process of truncation were applied to the complete name. In this way, as much as possible of the meanings of the names is preserved.

The Procedures Language provides a function (called REDUCE) which allows you to define the parameters for name reduction, in other circumstances.

## Description of the Name Reduction Process

The Name Reduction Process checks firstly 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, one of the following processes will occur:

- 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.

Secondly, if a name consists of 2 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 which is not a vowel
- If 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 2 characters.

If the number of constituent parts of a name is greater than the optimum number which would leave at least 2 characters in each part, separated by recognized separators, then the first of the following processes which will give the desired result will occur:

- 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 of Name Reduction**

A DB2 table, defined in the dictionary 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
- Two columns with identical names, which is illegal.

However, the Name Reduction Process would reduce the constituent parts of the first name:

- -SPECIAL,
- -ORDER
- -and MONTH

in turn, to achieve:

SPEC\_ORD\_DATE\_MONT

and reduce the constituent parts of the second name:

- SPECIAL
- and ORDER

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.



---

# Glossary

---

Below is a brief description of some of the terms used in this manual.

## **Array**

Consists of an identifiable set of elements where each element represents an item of data.

## **ASG Defined Variables**

These are variables which have names defined by ASG. They are:

- Local Variables (&L) in the range &L0 to &L99
- Global Variables (&G) in the range &G0 to &G9
- Installation Variables (&I) in the range &I0 to &I99
- Parameters (&P) in the range &P0 to &P99

There are also System Variables such as &CURS which are read-only variables that have values automatically assigned by Manager Products.

## **Command Variables**

These variables are defined by the COMMAND directive and assigned by an instruction of the form:

variable-name = expression

They retain their value for the duration of that Executive Routine including any called or calling Executive Routines. Unlike Global Variables the values assigned to Command Variables are not retained to the end of the logon session.

## **Corporate Executive Routine**

A type of Executive Routine that can only be set up by the Systems Administrator. They can be used, subject to Access Control, by all users.

## **Current Line**

A designated line of a ASG-ControlManager buffer created during a ASG-ControlManager interactive session. In the current ASG-ControlManager Release this is the top line of the Data Area when the buffer is displayed on the screen. There is always a Current Line for the buffer being processed even when that buffer is not displayed on the screen (as can occur when the contents of a buffer are processed by an Executive Routine).

### **Cursor Spatial Commands**

Executive Routines of this type are used with the Cursor System Variable (&CURS). The command is usually entered by means of a PF key. The value of the Cursor System Variable is determined using the position of the cursor on the screen.

### **Data**

Information which is supplied to, or created within an Executive Routine, for later use within the Executive Routine. Data is stored in Executive Routines within variables.

### **Directives**

These are instructions which either control the instruction sequence within an Executive Routine, or perform operations on data which is either internal to the Executive Routine or supplied by the user in the form of parameters, for example GOTO, IF, and WRITEL.

### **Executive Commands**

They are the same as Primary Commands in that they may operate on external data or parameters supplied by the user. However they may not be executed outside Executive Routines as they are only applicable within the context of Executive Routines. For example: DACCESS, SENDF, DRETRIEVE.

### **EXECUTIVE members**

These members reside in the MP-AID and can be used, subject to Access Control, by all users.

### **Executive Routine**

A generic term for Corporate Executive Routine, User Executive Routine, or Transient Executive Routine.

### **EXECUTIVE-ROUTINE members**

These members reside in the Manager Products Administration Dictionary. Once they have been constructed by the Systems Administrator onto the MP-AID, they become EXECUTIVE members on the MP-AID.

### **Full Evaluation**

Certain expressions, for example those given following a SAY directive are subject to Full Evaluation. Unlike Limited Evaluation arithmetic expressions and functions are evaluated.

### **Function**

An expression which returns a result, for example LENGTH(DRINK) will return a value of 5 indicating that the length of character string DRINK is 5 characters.

### **Global/Command User Defined Variable Index**

This index contains a record of all user defined Global or Command Variables which are currently active. Variables are set up in the index using the GLOBAL or COMMAND directives. Variables are removed from the index using the DROP or RELEASE directives.

### **Global Variables**

These variables are assigned by using an instruction of the form:

variable-name = expression

They retain their value for the duration of the Manager Products session, or until reassigned. They may have ASG defined names in the range &G0 to &G99 or they may have user defined names of up to 50 characters providing that the variable has been previously declared as global using the GLOBAL directive.

**Installation Variables**

These are variables which are assigned by the Systems Administrator during logon to Manager Products and cannot be reassigned by the general user.

**Instruction**

In an Executive Routine an instruction specifies an action which the Manager Products software is to perform. This comprises all components of an Executive Routine except for labels and comments. Instructions may be any of the following:

- Directives
- Primary Commands
- Executive Commands.

**Limited Evaluation**

Certain expressions, for example those given following a WRITEL directive are subject to Limited Evaluation. This differs from Full Evaluations in that arithmetic expressions and functions are not evaluated.

**Linear Commands**

Executive Routines of this type are normally used with parameters, and are input in the Line Command Area. The parameter values are derived from the contents of the associated data line.

**Local Variable Index**

This index contains a record of all Local Variables which are currently active. Variables are set up in the index using the LOCAL directives or when they are first assigned a value within the current Executive Routine. Variables are removed from the index using the DROP or RELEASE directive.

**Local Variables**

These variables are assigned by using an instruction of the form:

variable-name = expression

and retain their value only for the duration of the Executive Routine in which they are assigned or until reassigned. They may have ASG defined names in the range &L0 to &L99 or they may have a user defined name of up to 50 characters. User defined local variables must be declared as local using the LOCAL directive if the same variable name is currently in use as a Global or Command User Defined Variable.

**Manager Products Administration Dictionary**

A dictionary controlled by the Systems Administrator. This dictionary is used to set up EXECUTIVE-ROUTINE members.

**MP-AID**

The Manager Products Administrative and Information Dataset.

### **Parameters**

In the context of Procedures Language, parameters represent values which are input by the user at the time an Executive Routine is invoked.

### **Primary Commands**

These commands may operate on data which is either supplied by the user or is external to that generated by the Executive Routine. Primary Commands may be issued outside Executive Routines in the Command Line or within Executive Routines. For example:

```
LIST ONLY VERSION;
```

### **Systems Administrator**

The Systems Administrator, through the Manager Products Administration Dictionary and the MP-AID, controls access to all Manager Products and dictionaries, and is responsible for the configuration of the Manager Products environment.

### **System Assigned Variables**

System Variables are referred to as being System Assigned Variables as unlike all other types of variable their values cannot be assigned by users; they are automatically assigned by ASG-ControlManager.

### **System Variables**

These are variables which are automatically assigned by ASG-ControlManager and cannot be changed by any user.

### **Transient Executive Routine**

This type of Executive Routine is set up by a user as a TRANSIENT member.

### **TRANSIENT members**

Members of this type reside in the MP-AID. They are entered directly into the MP-AID by individual users. TRANSIENTs are automatically deleted when the originating user logs off from Manager Products.

### **User Assigned Variables**

User Defined Variables can be subdivided into variables which cannot be assigned by the user (System Assigned Variables, that is System Variables) and variables which may be assigned by the user (User Assigned Variables, that is all other variable types).

### **User Defined Functions**

Functions which may be written by users in a language other than the Procedures Language.

### **User Defined Variables**

These are variables which have user defined names of up to 50 alphanumeric characters long.

### **User Executive Routine**

This type of Executive Routine is set up by a user as a USER-MEMBER. It can only be accessed by the originating user or a user having the same Logon Identifier as the originating user.

**USER-MEMBERS**

Members of this type reside in the MP-AID. They are entered directly into the MP-AID by individual users, and can be accessed only by the originating user or a user having the same Logon Identifier as the originating user.

**Variables**

Locations to which names are assigned which allow data to be stored within Executive Routines.



## Symbols

- \* symbol 145
- ? symbol 145
  - in the EXTRACT command 145
  - in the ISQL command 150

## A

- ACQUIRE keyword 172
- ACTIVATE clause 177–179
- ADD clause 175–177
- ADDING keyword 160
- ALL clause 178
- ALL keyword 162
- ALTER keyword 173
- APPEND keyword 278
- AS clause 277
- Assembler SQL/DS host language data
  - structure generation 237

## B

- BOTH keyword 164
- BUFFER keyword 151

## C

- CARDINALITY clause 253
- children SQL/DS objects 145
- cluster diagrams
  - example of SQL PLOT CLUSTER
    - output 59–62
  - format of SQL PLOT CLUSTER
    - output 59
  - information given in SQL PLOT CLUSTER
    - output 60
- COBOL SQL/DS host language data
  - structure generation 237
- COLUMNS clause
  - SQL-TABLE 247
  - with the SQL ALTER command 175
- COMMAND members 115
- CONSTRAINT clause
  - SQL-TABLE 252

- with the SQL ALTER command 176, 178
- CONTAINS clause
  - SQL-INDEX 193
  - SQL-TABLE 248
  - SQL-VIEW 264
- conventions page vi
- Corporate Executive Routines
  - description of
    - MPDYMMLOCC 131
    - MPDYWTCVDT 130
    - MPDYWTDFLT 130
    - MPDYWTEXCC 131
    - MPDYWTMT42 131
    - MPDYWTOCOD 131
    - MPDYWTRDMR 130
  - SQL/DS dictionary definitions 130
- CREATE keyword in the SQL SYNONYM command 244
- CREATOR clause 144
- CREATOR-OWNER clause
  - SQL-DBSPACE 184
  - SQL-INDEX 193
  - SQL-TABLE 246
  - SQL-VIEW 263

## D

- database design
  - creating entity and userview
    - models 29
  - examples of generated definitions
    - SQL-INDEX 78
    - SQL-TABLE 77
    - SQL-VIEW 79
    - SYSTEM 80
  - examples of the design process 33
  - features to support SQL/DS 28
  - generated definitions
    - SQL members 76–80
    - SQL-INDEX 77
    - SQL-TABLE 76
    - SQL-VIEW 78

- SYSTEM 79
  - generating relational schema in the WBDA 29
  - generating SQL design in the WBDA 30
  - introduction 25
  - introduction to examples 33
  - introduction to referential structures and cycles 27
  - introduction to SQL PLOT CLUSTER 200
  - introduction to SQL REPORT 240
  - naming relations 29
  - overview 25
  - overview diagram of 23
  - populating dictionary with SQL member definitions 31
  - previewing member definitions 32
  - reporting SQL design 31
  - SQL objects generated from relational schema 30
  - SQL PLOT CLUSTER
    - all tables 201
    - command 200
    - tables in a specific format 202
    - tables selected by name 201
    - tables selected by WBDA number 201
  - SQL REPORT
    - all tables 240
    - command 240
    - tables in a specific format 242
    - tables selected by name 241
    - tables selected by WBDA number 241
  - support for referential integrity 26
  - the design process 29
  - DBSPACE keyword 144
  - DEACTIVATE clause in the SQL ALTER command 177–179
  - design process
    - examples 33
    - generating relational schema in WBDA 29
    - generating SQL design in the WBDA 30
    - introduction to examples 33
    - naming relations 29
    - populating dictionary with SQL member definitions 31
    - previewing member definitions 32
    - reporting SQL design 31
    - SQL objects generated from relational schema 30
    - the SQL/DS database design process 29
  - DETAILS keyword 164
  - DROP clause in the SQL ALTER command 175
  - DROP keyword in the SQL SYNONYM command 187, 245
  - DUPLICATES keyword 162
  - dynamic SQL services
    - an illustration 105
    - creating and populating a table 115
    - creating your own HELP text 122
    - importing information and assigning it to command variables 118
    - inserting rows into a table 117
    - submitting any SQL statement that can be prepared 120
- E**
- EXECUTIVE members 115
  - executive routines 109
  - EXPAND clause
    - nested GROUP-SQL-TABLE 250
    - SQL-INDEX 194
    - SQL-TABLE 249
    - SQL-VIEW 265
  - export
    - a table layout 237
    - defining the SQL/DS data type of columns 84
    - diagram of SQL statement and SQL/DS host language data structure generation 95
    - generating in host language
      - an SQL/DS host language data structure 237
      - an SQL/DS host language indicator structure 103
    - tailored DATE and TIME character field lengths structures 104
    - tailored SQL statements 97
  - SQL ACQUIRE statements 172
    - to activate or deactivate a primary key and all foreign keys 178
    - to activate or deactivate a primary key on a table 177
    - to add columns to a table 175
    - to add or drop a primary key on a table 175

- to add or drop a referential constraint on a table 176, 178
    - to combine alterations to a table 179
  - SQL ALTER TABLE statements 173
  - SQL COMMENT ON statement 180
  - SQL CREATE statement 182
  - SQL CREATE SYNONYM statement 244
  - SQL DROP statement 187
  - SQL DROP SYNONYM statement 244
  - SQL GRANT statement 190
  - SQL LABEL ON statement 180
  - SQL REVOKE statement 190
  - SQL/DS data types 96
  - EXTRACT SQL command 143
- F**
- FOR-BIT-DATA keyword 251
  - FOREIGN-KEY clause 253
  - FROM keyword 151
- G**
- GRANTOR clause 230
  - group
    - keyword in the RECONCILE command 162
    - SQL/DS column definition 84
- H**
- HELP keyword 150
- I**
- IGNORING clause 160
  - import 123
  - IMPORT keyword 154
  - IN clause 247
  - INDEX keyword 144
  - INITIALIZE keyword 158
  - ISQL command 149
  - ITEM members in SQL/DS column definition 84
- L**
- LIST keyword 164
  - LOCK clause 185
- M**
- MEMBER keyword 162
- N**
- name reduction process
    - description 279
    - example 280
    - introduction 279
  - NAMED clause 176, 178
  - names of members documenting imported SQL/DS objects 126
  - NEW keyword 278
  - NHEADER clause 185
  - NO-COMMON-CLAUSES keyword 163
  - NO-IMPACT keyword 189
  - NOPRINT keyword 278
  - NOT-NULL keyword 251
  - NO-XREF keyword 164
  - NUMBER clause 176, 178
  - NUMBER keyword 161–163
- O**
- ON clause 193
  - ONTO clause 278
- P**
- PAGES clause 185
  - parent SQL/DS objects 145
  - PCTFREE clause
    - SQL-DBSPACE 185
    - SQL-INDEX 195
  - PCTINDEX clause 185
  - PL1 SQL/DS host language data structure generation 237
  - POPULATE command 151
  - PREVIEW command
    - description of MPDYMMLOCC 131
    - for SQL/DS dictionary definitions 130
    - previewing the definitions generated from the WBTA 154
    - syntax 155
  - PRIMARY-KEY keyword
    - SQL ALTER 175, 177
    - SQL-TABLE 251
  - PRINT keyword 278
  - PRIVATE keyword
    - SQL-DBSPACE 184
    - with the SQL and PREVIEW IMPORT commands 278
  - privilege types
    - SQL-PRIVILEGE
      - ALL 231
      - PROGRAM 232
      - SYSTEM 232
      - TABLE 232

procedures language terminology 283  
PRODUCE keyword 237  
PROGRAM privilege 232  
PUBLIC keyword  
    SQL-DBSPACE 184  
    with the PREVIEW IMPORT  
        command 278

**R**

RADD command 156  
RECONCILE command 157  
    description of  
        MPDYMMLOCC 131  
        MPDYWTCVDT 130  
        MPDYWTDFLT 130  
        MPDYWTEXCC 131  
        MPDYWTMT42 131  
        MPDYWTOCOD 131  
        MPDYWTRDMR 130  
    syntax 168  
referenced SQL/DS objects 145  
RENAMING keyword 161  
REPLACE keyword 278  
REPLACING keyword 161  
RIGN command 168  
ROLLBACK keyword 152  
RREN command 169  
RREP command 170  
RUPD command 171

**S**

SIZE keyword 243  
SQL ACQUIRE  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    statement generation 172  
SQL ALTER command  
    ACTIVATE, DEACTIVATE  
        ALL clauses 178  
        CONSTRAINT,NAMED,  
            NUMBER clauses 178  
        PRIMARY-KEY clauses 177  
    ADD COLUMNS 175  
    ADD CONSTRAINT NAMED and  
        NUMBER 176  
    ADD PRIMARY-KEY 175  
    combinations of changes 179  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100

DROP PRIMARY-KEY 175  
    syntax 179  
SQL cluster plot  
    example 59  
    format of cluster diagram 57  
    information in cluster diagrams 58  
    introduction 56  
    SQL design relationship matrix 62  
SQL COMMENT command  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    generating SQL CREATE,  
        COMMENT ON, and LABEL  
        ON statements 102  
    syntax 181  
SQL CREATE command 182  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    generating SQL CREATE,  
        COMMENT ON, and LABEL  
        ON statements 102  
    syntax 183  
SQL CREATE SYNONYM statement  
    generation 244  
SQL DROP command  
    an example of the impact analysis  
        report 189  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    impact analysis report 188  
    NO-IMPACT keyword 189  
    syntax 190  
SQL DROP SYNONYM statement  
    generation 244  
SQL GRANT command 190  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    syntax 191  
SQL LABEL command 180  
    deriving SQL/DS object names from  
        aliases 101  
    displaying internal diagnostic  
        output 100  
    generating SQL CREATE,  
        COMMENT ON, and LABEL  
        ON statements 102

- syntax 182
- SQL LIST CYCLES 197
  - output
    - description 75
    - example 75
    - introduction 74
  - syntax 198
- SQL LIST TABLES command 198
  - output
    - description 73
    - example 74
    - introduction 73
  - syntax 200
- SQL PLOT CLUSTER command 200
  - cluster plot
    - tables in a specific format 202
    - tables selected by WBDA number 201
  - output
    - example 59
    - for all tables 201
    - format of cluster diagram 57
    - information in cluster diagram 58
    - introduction 56
    - SQL design relationship matrix 62
    - tables selected by name 201
  - syntax 202
- SQL PLOT REFERENTIAL-STRUCTURES command 203
  - displaying
    - all referential structures 204
    - one referential structure 206
  - output
    - example 71
    - introduction 64
    - layout 65
  - syntax 207
- SQL POPULATE command 208
  - combining options 216
  - examples 216
  - generating
    - SQL-INDEX members 211
    - SQL-TABLE members 209
    - SYSTEM member 213
  - introduction 208
  - populating SQL-VIEW members 212
  - references
    - to dbspaces 210
    - to SQL-USER members 213
  - selecting tables in Workbench Design Area 214
  - support for referential integrity 209
  - suppressing support for referential integrity 210
  - syntax 217
  - tailoring generated definitions 215
- SQL PREVIEW command 218
  - combining options 227
  - examples 227
  - generating
    - SQL-INDEX definitions 222
    - SQL-TABLE definitions 220
    - SQL-VIEW definitions 223
    - SYSTEM definition 224
  - references
    - to dbspaces 221
    - to SQL-USER members 224
  - selecting tables in the Workbench Design Area 225
  - support for referential integrity 220
  - suppressing support for referential integrity 221
  - syntax 228
  - tailoring generated definitions 226
- SQL PRODUCE command 236
  - deriving external names from aliases 101
  - displaying internal diagnostic output 100
  - generating a host language indicator structure 103
  - syntax 239
  - tailoring DATE and TIME character field lengths 104
- SQL REPORT command 240
  - output
    - contents of tables 48
    - example 50
    - foreign key relationships 49
  - report tables
    - ALL 240
    - in a specific format 242
    - selected by name 241
    - selected by WBDA number 241
  - syntax 242
- SQL REVOKE command 242
  - deriving SQL/DS object names from aliases 101
  - displaying internal diagnostic output 100
  - syntax 191
- SQL SIZE command 243
  - displaying internal diagnostic output 100

- syntax 244
  - SQL SYNONYM command 244
    - deriving SQL/DS object names from
      - aliases 101
    - displaying internal diagnostic output 100
    - syntax 245
  - SQL table report 48
    - contents of tables 48
    - example 50
    - foreign key relationships 49
    - introduction 48
  - SQL-COMMENT clause 180
    - SQL-TABLE 254
    - SQL-VIEW 268
  - SQL-DBSPACE 184
    - example 186
    - introduction 184
    - syntax 187
  - SQLI\_CD\_n variable 114
  - SQLI\_CL variable 114
  - SQLI\_COMMAND variable 113
  - SQLI\_CS variable 114
  - SQLI\_RETURN variable 114
  - SQLI\_ROWS variable 114
  - SQLI\_SQLCODE(1) variable 114
  - SQLI\_SQLCODE(2) variable 114
  - SQLI\_TABLE\_NAME variable 113
  - SQLI\_TABLE\_SPACE variable 113
  - SQL-INDEX 192
    - example 196
    - introduction 192
    - syntax 196
  - SQL-LABEL clause 197
    - SQL-TABLE 254
    - SQL-VIEW 268
  - SQL-PRIVILEGE 229
    - examples 233
    - introduction 229
    - syntax 235
  - SQL-TABLE 245
    - examples 255
    - introduction 245
    - syntax 257
  - SQL-USER 259
    - example 261
    - introduction 260
    - syntax 261
  - SQL-VIEW 262
    - examples 269
    - introduction 262
    - syntax 275
  - STORPOOL clause 186
  - SUMMARY keyword 164
  - SYNONYMS clause 260
- T**
- TABLE keyword 144
  - TABLE privilege 231
  - TABLE-LAYOUT keyword 237
- tailoring
- export 97
  - help text 122
  - import 128
- U**
- UNIQUE keyword 193
  - USING clause
    - in the PREVIEW command 154
    - in the RECONCILE command 159
- V**
- VIEW keyword 144
- W**
- WBDA number 201



ASG Worldwide Headquarters Naples Florida USA | [asg.com](http://asg.com)