IBM VisualAge DataAtlas Multiplatform

Dictionary and Designer User's Guide

Version 2.5

SC26-9134-00

IBM VisualAge DataAtlas Multiplatform

Dictionary and Designer User's Guide

Version 2.5

SC26-9134-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vii.

First Edition (September 1997)

This edition applies to IBM VisualAge DataAtlas Multiplatform, Version 2.5, Program Number 5648-A48, and to any subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality.

If you have comments about this publication, please go to the DataAtlas Web site, select “Browse or participate in the DataAtlas threaded discussion group,” and add a comment to the “Online Information and Publications” thread. The Web site is located at http://www.software.ibm.com/ad/datatlas.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices . . . . . **vii**
Trademarks vii

About This Book . . . . . **ix**
How This Book Is Organized. ix
Related Publications. x
DataAtlas Web Site x
Terminology Note x

Summary of Changes . . . . . **xi**

Part 1. Dictionary User's Guide . . **1**

Chapter 1. Introduction to DataAtlas Dictionary **3**
DataAtlas Dictionary Functions 4
Populating 4
Reconciling. 5
Creating and Deleting 5
Updating 6
Generating 6
Querying 6
Becoming Familiar with the Interface . . . 7
Using Online Information 8
The Online Tutorial 8
Online Help 8

Chapter 2. Getting Started **9**
Launching DataAtlas Dictionary and Designer 9
DataAtlas and TeamConnection 10
TeamConnection Overview 10
How Workfolders Relate to TeamConnection 11
The Profile Notebook 11
DataAtlas Naming Scheme 12

Chapter 3. Searching for Objects in the TeamConnection Database **15**
Creating a Workfolder 15
Searching for Objects from a Workfolder . . 15

Chapter 4. Creating and Deleting Objects . **19**
Creating Relational Database Objects . . . 19
Creating IMS Objects 21
Creating Included Source Definitions . . . 22
Creating Shareable Data Elements 24
Creating Shareable Data Structures. . . . 24
Deleting Objects 27

Chapter 5. Populating the TeamConnection Database **29**
Populate Considerations When Using DataAtlas Modeler or Designer 30
Relational Design 30
Physical Design 30
Populating the TeamConnection Database from DB2 Catalogs 31
Using a Reconcile Mapping Table during DB2 Populate 33
Populating IMS Definitions 33
Using a Reconcile Mapping Table during IMS Populate 35
Populating COBOL and PL/I Data Structures 35
Using a Reconcile Mapping Table with COBOL or PL/I 37
Running a Trial Run 37
Reconciling Data Definitions during Populate 38
Creating Shareable Data Components without Reconciling. 39
Mapping Tables 39
Structure of a Mapping Table 39
Examples of Entries. 41
Creating a Mapping Table from a Prototype 41

Chapter 6. Updating Objects **43**
Updating Relational Database Objects. . . 43
Updating IMS DBD Objects 45
Updating IMS PSB Objects 46
Updating COBOL or PL/I Included Source Definitions 47
Updating Shareable Data Structures . . . 49

Chapter 7. Generating Definitions . . . **53**

Generating Definitions from DataAtlas . . 53
HLL Cross-Generation 54
Generating Definitions with a Build Script . 55
Using DataAtlas DDL on a DB2/400 . . . 55

Chapter 8. Running TeamConnection SQL Queries 57
Running a Query. 57
Supplied Queries. 57
Running a Supplied Query 58
Writing TeamConnection SQL 61
The Department-Employees Example . . 61
Advantages of Querying an Object Data Model 63
Using Qualifiers with Nested Collections. 63
Querying across Multiple Object Classes . 64
Queries with an Outer Join 64
TeamConnection's Views and Attributes . . 65

Part 2. Designer User's Guide . . 67

Chapter 9. Introduction to DataAtlas Designer 69

Chapter 10. Database Design Concepts . 71
The Central Concept: The Table 71
Relational Design 71
Physical Design 72
Data Load and Work Load 72
Storage 73
Access 74
Knowledge of Database Design 74

Chapter 11. Database Design with DataAtlas Designer 75
Design Modes. 75
Design Using Notebooks 75
Design Using Design Support 75
Design Areas 77
Designing Single Objects 77
Designing a Set of Objects. 78
Design Information 78
Design Reports 79
Design Scenarios 79
Creating a New Database 80
Optimizing an Existing Database . . . 80

Chapter 12. Using DataAtlas Designer . . 83
Configuring the Design Environment . . . 83
Requesting Design Support from a Physical Design 84
Requesting Design Support from a Workfolder. 85
Designing Database Objects Using Notebooks 86
Designing by Tuning an Object's Definition 86
Requesting Design Support from a Notebook 88
Selecting Design Actions 88
Requesting Design Reports 90
Evaluating Design Support Reports . . 90
Accepting Design Proposals 92
Selecting from a Choice of Proposals . . 93
Executing Design Proposals 93

Chapter 13. Designing Tables 95
Available Design Support for Tables . . . 96
Specifying General Information 96
Viewing the Shareable Table Definition . . 97
Viewing the Columns 97
Adding and Deleting Columns 97
Optimizing the Table Layout 98
Assigning the Table to a Physical Design . . 98
Specifying Routines and Options 99
Creating and Modifying a Primary Key . . 99
Creating and Modifying Unique Keys. . . 100
Creating and Modifying Foreign Keys. . . 101
Creating Indexes 102
Assigning the Table to a Table Space and a Database 103
Creating Table Partitions 103
Modifying Table Partitions 104
Specifying the Data Load 105
Specifying the Work Load. 105
Extracting DB2 Actual Values 106

Chapter 14. Designing Columns 107
Available Design Support for Columns . . 107
Defining a Column 108
Setting Options 109
Specifying the Data Load 109
Specifying the Work Load. 110
Viewing DB2 Actual Values 110

Chapter 15. Designing Indexes 111
Available Design Support for Indexes . . . 111

Specifying General Information 112
Assigning the Index to a Physical Design . 112
Defining an Index 112
Specifying Index Columns 113
Specifying the Sorting Order 113
Specifying Storage Information 114
Viewing Index Partitions 114
Specifying Type and Storage Information of an Index Partition 114
Viewing DB2 Actual Values 115

Chapter 16. Designing Table Spaces . . 117
Available Design Support for Table Spaces . 117
Specifying General Information 118
Assigning the Table Space to a Physical Design 118
Assigning the Table Space to a Database . . 119
Setting Options 119
Creating Table Space Partitions 119
Specifying Type and Storage Information of a Table Space Partition 120
Specifying Storage Information 120
Assigning Tables 121
Selecting a Buffer Pool 121
Specifying Design Information 121
Extracting DB2 Actual Values 121

Chapter 17. Designing Databases . . . 123
Available Design Support for Databases . . 123
Specifying General Information 124
Setting Options 124
Specifying Design Information 124
Viewing Assigned Table Spaces 124
Assigning the Database to a Storage Group . 125
Selecting a Buffer Pool 125
Assigning the Database to a Physical Design 125

Chapter 18. Designing Storage Groups . 127
Available Design Support for Storage Groups 127
Specifying General Information 128
Assigning the Storage Group to a Physical Design 128
Specifying Storage Information 128
Assigning Databases 129
Viewing Assigned Table Spaces 129
Viewing Assigned Indexes 129
Specifying the Usage Intent 130
Specifying the Required Space 130
Converting the Used Space Value 130
Viewing DB2 Actual Values 131

Chapter 19. Designing Views 133
Specifying General Information 133
Defining a View 133
Assigning the View to a Physical Design . . 134

Chapter 20. Designing an Alias/Synonym . 135
Specifying General Information 135
Defining an Alias/Synonym 135
Assigning the Alias/Synonym to a Physical Design 136

Appendix A. Sample Files Shipped with DataAtlas 137
DB2 UDB Sample Tables 137
IMS Sample Files. 138
COBOL Sample Files 138
PL/I Sample Files 139
Sample Reconcile Mapping Tables 139
Sample COBOL Command File 139
Sample PL/I Command File 139

Appendix B. Qualifiers for Object Names. 141
Relational Qualifiers. 142
IMS Qualifiers. 143
Prefix Qualifiers 144

Appendix C. Using the DATATLAS.EXE Build Script 145
Sample Build Script 145
Input Parameters. 145
TeamConnection Build 146
TeamConnection DataAtlas Objects. 146
DataAtlas DB2 Sample Script Settings . . . 147
Input Parameters. 147
Special Considerations 148
Return Codes 148
DataAtlas Oracle Sample Script Settings . . 149
Input Parameters. 149
Return Codes 149
Special Considerations 149
DataAtlas IMS Sample Script Settings . . . 150
Input Parameters. 150
Return Codes 150
Special Considerations 150
DataAtlas COBOL Sample Script Settings . 150

Input Parameters 150
Return Codes 151
Special Considerations 151
DataAtlas PL/I Sample Script Settings . . 151
Input Parameters 151
Return Codes 152
Special Considerations 152

Appendix D. TeamConnection Considerations 153
Renaming Objects 153
Workfolders and the TeamConnection Cache 153
TeamConnection Locking 154
Concurrent Versus Serial Development . . 155

Appendix E. PL/I Supported Data and Nondata Attributes 157

Appendix F. Views and Attributes of Object Types 161

Appendix G. Actions and Rules 211
Object Types, Functions, and Actions . . . 211
Rule Explanations 214

Appendix H. Performance Considerations 225
Populating 225
Maintenance 225
Queries 226
Designer 226
Modeler 226

Glossary 227

Index 229

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department W92/H3, P.O. Box 49023, San Jose, CA 95161-9023. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

DataAtlas	IMS
DataGuide	OS/2
DB2	OS/390
DB2 Universal Database	TeamConnection
Electronic Showcase	Visual Age
IBM	

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Other company, product, and service names may be trademarks or service marks of others.

About This Book

Read this book to find out how to use the Dictionary and Designer components of IBM VisualAge DataAtlas Multiplatform. The Dictionary component, *DataAtlas Dictionary*, is an advanced data dictionary that enables you to populate, create, update, and generate data definitions for database management systems and high-level languages. The Designer component, *DataAtlas Designer*, is a set of design functions that checks the correctness and completeness of relational database objects, proposes design improvements, and implements the proposals at your request.

This book assumes you have installed and started DataAtlas on your desktop. Instructions for installing and starting DataAtlas are under the "Installation" topic on the CD-ROM's Electronic Showcase.

You should know how to work with objects in OS/2 or in Windows NT. You should also be familiar with the concepts of your installed database management system, and how to use it.

How This Book Is Organized

This book has two parts and a set of appendixes with supplemental information. Part 1 is a guide to using DataAtlas Dictionary. It introduces DataAtlas Dictionary and leads you through scenarios for these tasks:

- Searching for objects in the TeamConnection database
- Creating and deleting objects in the TeamConnection database
- Populating the TeamConnection database
- Updating objects in the TeamConnection database
- Generating definitions from the TeamConnection database
- Running TeamConnection SQL queries

Part 2 is a guide to using DataAtlas Designer. It introduces DataAtlas Designer, explains design concepts, and describes the interface for using the design functions. Its chapters show you how to use the design functions for each of these DB2/390 objects: tables, columns, indexes, table spaces, databases, storage groups, views, and alias or synonyms.

Related Publications

DataAtlas has a third component, *DataAtlas Modeler*. It's a data modeling tool that supports the entity-relationship approach to conceptual data modeling. To find out how to use it, read *Modeler User's Guide*, SC26-9041.

For general information on how to install DataAtlas, refer to *Installation Overview*, SC26-9042. This booklet is supplemented by instructions under the "Installation" topic on the CD-ROM's Electronic Showcase.

For information about using TeamConnection, refer to *TeamConnection User's Guide*, SC34-4499.

To order these publications, contact your local IBM representative. In the United States, you can order these publications directly by calling 1-800-879-2755.

DataAtlas Web Site

The DataAtlas Web site provides the latest product information. It also lets you communicate directly with members of the product development team and provides product service updates.

If you have comments about this publication, please go to the DataAtlas Web site, select "Browse or participate in the DataAtlas threaded discussion group," and add a comment to the "Online Information and Publications" thread. The Web site is located at *http://www.software.ibm.com/ad/datatlas*.

Terminology Note

This book and the DataAtlas user interface contain two repeatedly used abbreviations that may be unfamiliar to you:

- DB2/390, which refers to any of these DB2 enterprise data servers: DB2 Version 3, DB2 for MVS Version 4, and DB2 for OS/390 Version 5.
- DB2 UDB, which refers to DB2 Universal Database Version 5.

Summary of Changes

This book contains information that was formerly in two DataAtlas publications: *Dictionary User's Guide* and *Designer User's Guide*. The merged information has been modified to reflect these enhancements to Version 2.5 of IBM VisualAge DataAtlas Multiplatform:

- **Support for the latest DB2 products**

  DataAtlas now supports DB2 Universal Database. The abbreviation "DB2 UDB" is used in task scenarios to identify objects associated with this product.

- **Support for generating PL/I include files**

  "Chapter 7. Generating Definitions" on page 53 shows that the Generate function can now export PL/I include files.

  "Appendix C. Using the DATATLAS.EXE Build Script" on page 145 gives instructions for generating PL/I include files using the DATATLAS.EXE build script.

- **Integration with Component Broker**

  "Launching DataAtlas Dictionary and Designer" on page 9 describes an enhancement to DataAtlas's command interface that enables Component Broker and DataAtlas to communicate efficiently.

- **Addition of prototype mapping tables**

  "Creating a Mapping Table from a Prototype" on page 41 explains what prototype mapping tables are, how to create them, and how they help you quickly produce mapping tables to use in reconcile operations.

- **Extended design support for DB2 for OS/390 Version 5**

  When DataAtlas Designer proposes and validates Version 5 table space definitions, it ensures they agree with index definitions. The new design rules that provide this support are explained in "Appendix G. Actions and Rules" on page 211.

Along with these changes, the book's appendixes have been extended:

- "Appendix D. TeamConnection Considerations" on page 153 has been supplemented with new information.
- "Appendix F. Views and Attributes of Object Types" on page 161 has been extended to include entries for IMS object types.
- "Appendix H. Performance Considerations" on page 225 is a new appendix that will help you optimize the performance of DataAtlas.

Part 1. Dictionary User's Guide

Chapter 1. Introduction to DataAtlas Dictionary

DataAtlas Dictionary gives you a method to control, share, and standardize the data definitions associated with relational databases, IMS databases, and high-level language (HLL) applications. DataAtlas Dictionary uses TeamConnection for storing, maintaining, and sharing these data objects with DataAtlas Modeler and Designer, and with other tools in IBM's application development environments.

A primary mechanism for controlling data definitions is through the use of shareable objects which can be used in many database and HLL applications. The shareable objects you can create and maintain with DataAtlas Dictionary are *shareable data elements*, *shareable data structures* and *shareable table definitions*. Shareable data elements specify the data characteristics of elementary pieces of information, such as a zip code or employee name. Shareable data structures collect and order a related set of elementary information to be used as a unit, such as street, city, state, zip code and country which together constitute an address. Shareable table definitions specify the layout of a relational database table, such as an employee record or inventory maintenance table. Shareable table definitions can include shareable data elements. Shareable data structures can include both shareable data elements and nested shareable data structures. *Shareable data components* is a term used throughout this book to refer to shareable data elements or shareable data structures.

TeamConnection allows you to control and monitor access to your shared objects. Using the component structure in TeamConnection, you can separate shareable and non-shareable objects, and control access to the objects. For more information about TeamConnection components, see *TeamConnection User's Guide*.

Version 2.5 of DataAtlas Dictionary supports:

- DB2/390 databases

  "DB2/390" refers to any of these DB2 enterprise data servers: DB2 Version 3, DB2 for MVS Version 4, and DB2 for OS/390 Version 5.
- DB2 UDB databases

  "DB2 UDB" refers to DB2 Universal Database Version 5.
- DB2 Common Server Version 2 databases

  DB2 Common Server isn't mentioned elsewhere in this book, but wherever support for DB2 UDB is discussed, support for DB2 Common Server is implied as well.

- Oracle Version 7.3 databases
- IMS full-function and Fast Path databases

  DataAtlas supports all the versions of IMS/ESA through Version 6.
- COBOL COPY files

  DataAtlas supports IBM VisualAge for COBOL for OS/2 and Windows Version 2.0; IBM VisualAge for COBOL, Professional for OS/2 Version 2.0; and IBM VisualAge for COBOL Version 1.2 Refresh.
- PL/I include files

  DataAtlas supports IBM PL/I for OS/2 Professional Version 1.2; and IBM PL/I for Windows Version 1.2. DataAtlas requires the compilers to be at the CSD#4 service level or higher.

DataAtlas Dictionary Functions

DataAtlas Dictionary lets you import data definitions from existing production environments into the TeamConnection database. You can also create data definitions directly by initializing settings notebooks.

Once a data definition is in the TeamConnection database, you can update it to suit your changing requirements. When you want to put a new or updated data definition back into your operating environment, you can generate a workstation file containing definition language that conforms to the definition in the TeamConnection database.

Here is a closer look at each of DataAtlas Dictionary's principal functions, all available via DataAtlas Dictionary's object-oriented graphical interface.

Populating

Populating is the process DataAtlas uses to create objects in a TeamConnection database. The objects are based on data definitions in DB2 and IMS databases. They can also be based on COBOL COPY files and PL/I include files; in this case, the populated objects are called *included source definitions*.

In populating DB2/390 and DB2 UDB tables, DataAtlas refers to the appropriate catalog and additionally populates the indexes, views, synonyms, aliases, table spaces, databases, and storage groups the tables need.

IMS data definitions are accessed from a workstation file, after they're downloaded; source code comments on the DBDs and PSBs are also populated into the TeamConnection database.

COBOL COPY files and PL/I include files are accessed from workstation files. COPY files used with another COBOL compiler such as COBOL for MVS & VM can be populated if the data descriptions can be compiled with the VisualAge for COBOL compiler.

If you plan to populate COBOL or PL/I data structures using DataAtlas, you need to install or have access to an installed copy of a compiler listed above.

The compilers must be installed with the **COBOL Compiler and Runtime Library** or **PL/I Compiler and Runtime Library** option selected. Additional compiler installation options may be selected, but they are not required by DataAtlas. Changes made to your OS/2 CONFIG.SYS file or Windows NT registry during the compiler installation process must be in effect when using the COBOL and PL/I Populate and Generate functions.

You can populate the TeamConnection database in a "trial run" manner, where the definitions are processed by DataAtlas but are not saved and a report is generated. After you are sure of the definitions, you can perform the actual populate.

IBM DB/DC Data Dictionary users can migrate existing data definitions to the TeamConnection database by:

1. Applying the fixes for APARs PN87038 and PQ07716
2. Using the dialogs supplied by these fixes to extract data
3. Running the extracted data through IBM VisualAge Exchange Version 2.5
4. Using the Import function of TeamConnection

Reconciling

When you populate the TeamConnection database, DataAtlas Dictionary gives you the ability to *reconcile* data definitions. This allows you to create and use shareable data components. Reconciling data definitions also prevents the creation of duplicate data definitions. When populating an object, you can create a new shareable data component or use a shareable data component that already exists in the TeamConnection database. If you don't reconcile the data, any new object that you populate is defined as a local data component, and won't be shareable among TeamConnection database objects.

Creating and Deleting

DataAtlas Dictionary makes it easy to create objects of these types:

- **DB2 UDB** - Tables, indexes, databases, table spaces, and views
- **DB2/390** - Tables, indexes, views, storage groups, synonyms, aliases, table spaces, and databases

- **Oracle** - Tables, indexes, table spaces, and views
- **IMS** - DBDs, PSBs, and PCBs
- **COBOL and PL/I** - Included source definitions
- **Shareable data elements**
- **Shareable data structures**

You can also delete an object from the TeamConnection database. Before deleting an object, DataAtlas Dictionary shows you how other objects will be affected, and gives you a chance to delete them as well.

Updating

With DataAtlas Dictionary, you can easily change an object's characteristics, such as its name, length, and connection to a shareable data element. DataAtlas Dictionary checks the contents of notebook fields to help prevent you from entering invalid data.

Generating

DataAtlas Dictionary generates source statements from data in the TeamConnection database. The Generate function is, more or less, the opposite of the Populate function; it exports COBOL COPY files, PL/I include files, DBD and PSB macro statements, and SQL DDL to workstation files. To run your generated output in an OS/390 environment, you must upload it to the host.

You can generate data definitions by using the DataAtlas Dictionary user interface or by using a build script. A sample build script, DATATLAS.EXE, is shipped with DataAtlas. See "Appendix C. Using the DATATLAS.EXE Build Script" on page 145 for more information.

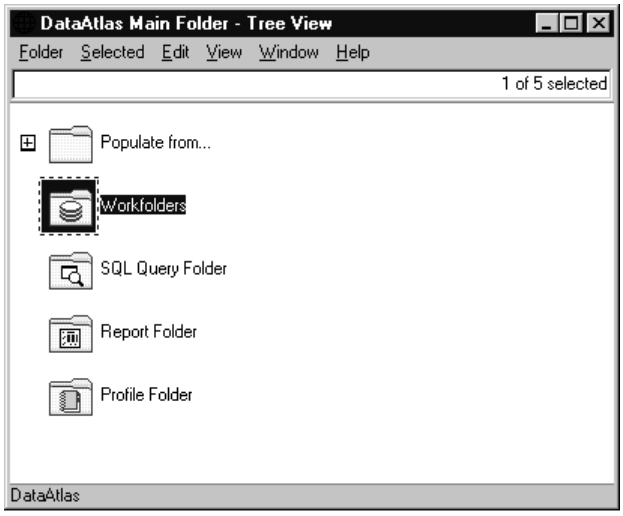
Querying

You can use TeamConnection SQL to run queries about the data definitions stored in the TeamConnection database. DataAtlas Dictionary provides many predefined queries, which you can modify, or you can create your own. TeamConnection SQL, based on an object data model, is slightly different from the standard SQL, which is based on a relational data model. For more information on TeamConnection SQL and the predefined queries, see "Chapter 8. Running TeamConnection SQL Queries" on page 57.

Becoming Familiar with the Interface

Before you begin to use DataAtlas Dictionary's functions, you need to be familiar with the Main Folder window. The Main Folder window opens when you start DataAtlas Dictionary and Designer.

Here's what the Main Folder window looks like:



The window consists of these parts:

- **The menu bar.** It shows the actions you can perform with DataAtlas.
- **Folders.** They represent the main DataAtlas functions.
- **The information line.** Located at the bottom of the window, the information line describes whatever menu-bar choice or icon you select.

In brief, here's what you'll find in the folders:

- **Populate from folder.** Objects inside this folder enable you to populate objects into the TeamConnection database from two sources: DB2 catalogs and workstation directories. For more information on populating objects into the TeamConnection database, see "Chapter 5. Populating the TeamConnection Database" on page 29.
- **Profile folder.** It contains an initial Profile notebook where you set default values for the DataAtlas Dictionary functions.
- **Workfolders folder.** Double-click this icon to see your *workfolders* or to create a new workfolder. A workfolder is a place for collecting objects that belong to the same *version*. A version is a given family, release, and work

area within TeamConnection. From DataAtlas Dictionary, the workfolder is the primary way to access data that is in the TeamConnection database. Once you have a workfolder, you can search for an object or update an object in the TeamConnection database. For more information on creating a workfolder and searching for objects, see “Chapter 3. Searching for Objects in the TeamConnection Database” on page 15.

- **Report folder.** It holds the results of DataAtlas query reports.
- **SQL Query folder.** A *query* is a request for information from the TeamConnection database based on conditions you specify through the command interface for SQL statements. For more information about queries, see “Chapter 8. Running TeamConnection SQL Queries” on page 57.

With your mouse pointer over any of the folders, click the right mouse button to see a pop-up menu of available actions. Double-click a folder to open it.

Using Online Information

DataAtlas provides online help and an online tutorial to help you learn about and maneuver through the product.

The Online Tutorial

Using the online tutorial and the sample files, you can do many of the DataAtlas Dictionary and Designer tasks described in this book and in *DataAtlas Designer User’s Guide*. To use the online tutorial:

1. From any window with a menu bar, click **Help**, then **Tutorial**.

   Result: The Tutorial window opens.
2. Take a few of the lessons to get “hands-on” experience with DataAtlas Dictionary.
3. When you’re done, double-click the small icon at the top-left corner of the Tutorial window.

   Result: A Close message box appears. Click **OK** to close the tutorial.

Online Help

You can get comprehensive online help on DataAtlas Dictionary and Designer tasks, concepts, terms, and windows from any window with a menu bar. Just click **Help** on the menu bar and select from the pull-down menu.

To get an explanation of the window you’re viewing, click **Help** on the window or put the mouse pointer anywhere in the window and press F1.

Chapter 2. Getting Started

Before you can begin using DataAtlas Dictionary, you must:

- Install TeamConnection and create a TeamConnection family.
- Create TeamConnection components to hold the DataAtlas Dictionary objects you will create.
- Create the TeamConnection release and work area that will be used to manage the versions of your DataAtlas objects.

To use DataAtlas with DB2 UDB or DB2/390 databases, you or someone with database administrator authority must issue the following commands in the `client` subdirectory (in OS/2) or `bin` subdirectory (in Windows NT) under the directory where DataAtlas is installed.

- To connect to the DB2/390 or DB2 UDB database issue:

  ```
  db2 connect to databasename
  ```

- To enable DataAtlas to populate data definitions from a DB2/390 or DB2 UDB database, issue:

  ```
  db2 bind ewsddb2x.bnd
  ```

  If you get authorization errors with this command, modify it by identifying the collection referenced in the bind file:

  ```
  db2 bind ewsddb2x.bnd collection atlasrdb
  ```

- To grant authority to users who will populate data definitions from existing DB2 databases and select from system catalog tables, issue:

  ```
  db2 grant bind, execute on package atlasrdb.ewsddb2x to userid
  ```

  where *userid* is the user ID of each user to be granted authority.

If a user does not already have connect authority to a DB2 database, you must issue the following commands:

```
db2 connect to databasename

db2 grant connect on database to userid
```

Launching DataAtlas Dictionary and Designer

You can launch DataAtlas Dictionary and Designer in either of two ways:

- Double-click the **DataAtlas Dictionary and Designer** icon on your desktop.

- Run the command `ewsuiv2.exe` in the `client` subdirectory (in OS/2) or `bin` subdirectory (in Windows NT) under the directory where DataAtlas is installed.

Using the `ewsuiv2.exe` command, you can bring up DatatAtlas Dictionary and Designer with the windows of your choice already open. To do so, add the `-file` *filename* parameters to the command. Statements in the named file identify the windows you want to be open when DataAtlas comes up. Each statement begins with `EWS_Open` and specifies the TeamConnection version, object type, and object name. For example:

```
EWS_Open, atlas|r|w, DSRMRDPhysicalDesign <>MyPhysDes<:>
```

This statement directs DataAtlas to open the physical design `MyPhysDes` in the Team Connection family `atlas`, release `r`, work area `w`. Component Broker uses this interface to communicate with DataAtlas and examine physical designs.

DataAtlas and TeamConnection

DataAtlas works with TeamConnection to provide a comprehensive means of maintaining complex databases. DataAtlas lets you create and update database objects; TeamConnection lets you track such changes in complex development and maintenance environments.

TeamConnection Overview

All TeamConnection activities are performed within a *version context*, which represents the family, release, work area, and component that your objects are related to. The first time you start DataAtlas Dictionary, you will be prompted to enter a default version context, which will be stored in your DataAtlas Dictionary profile and can be revised at any time. You must be authorized to a version context before that version context can be used to access the TeamConnection database. Your TeamConnection user ID is identified to TeamConnection either via the TC_USER environment variable on OS/2 or the Windows NT user name.

TeamConnection groups objects into units called components, which you must have authority to use. For example, a TeamConnection administrator may create a component for relational objects, one for IMS objects, and another for high-level language objects, and grant authority to different subsets of users. The authority level will allow you to perform all DataAtlas functions.

You won't normally run into complications when creating and updating objects within a single component; however, you should be aware of the implications of updating objects with relationships between components.

When a relationship is created between two objects in two different components, you are required only to have authority to update objects in the component where the source object resides. For example, you may wish to connect a column in the DB2 UDB table EMP_RECORD (in component RDB) with a shareable data element ZIPCODE (in component DE). You can do this if you have update authority for the RDB component; you are not required to have update authority for the DE component.

How Workfolders Relate to TeamConnection

DataAtlas workfolders represent TeamConnection work areas or, more specifically, a particular version within which changes are made. When a DataAtlas workfolder is created, it must be associated with a specific family, release, and work area combination. All activities that occur within that workfolder will occur within the version of TeamConnection represented by the workfolder. Changes to a workfolder, which would involve adding or removing objects that are being maintained, are stored within TeamConnection and managed by DataAtlas. Because of this, you should close a workfolder so that its contents can be saved before performing any work area activity from the TeamConnection tool. For example, before refreshing, freezing, or integrating a work area, all workfolders within that version should be closed.

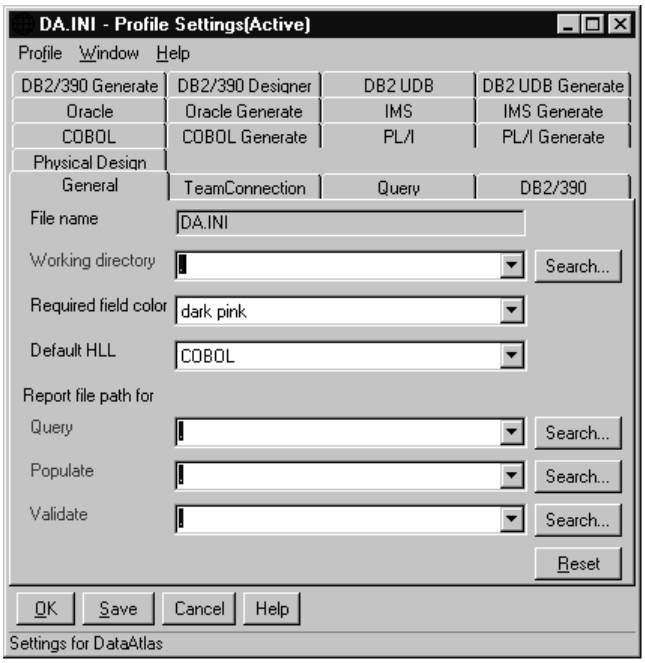
Multiple workfolders can co-exist in DataAtlas to represent different versions. All workfolders are grouped within the Workfolders folder on the DataAtlas Main Folder Window. You should keep the names of your workfolders unique across families and versions.

See “Appendix D. TeamConnection Considerations” on page 153 for more information on using DataAtlas with TeamConnection.

The Profile Notebook

You can use the Profile notebook to set up information such as working directory, report locations, and settings to use when creating objects in the TeamConnection database.

To open a Profile notebook, double-click the **Profile** folder in the DataAtlas Main Folder window. Then double-click the icon to open your Profile notebook. Your Profile notebook looks like this:



Looking through its settings pages, you see references to DataAtlas functions that aren't in the Main Folder window. For example, on the **General** page, you can specify the path where you'd like the reports from Query, Populate, and Generate to reside.

The **TeamConnection** page is a place for identifying the default values (family, release, and work area) you're working with. You can override the default values. Your TeamConnection data administrator can advise you about these settings.

Most of the remaining pages contain settings to be used when a DB2, Oracle, IMS, COBOL, or PL/I object is populated into or generated from the TeamConnection database.

DataAtlas Naming Scheme

All DataAtlas objects adhere to a consistent naming scheme, which makes it easy to locate and identify objects in any DataAtlas component.

Each DataAtlas object has a multipart name of the form:

```
<prefix>access-name<variation:revision>
```

prefix The foremost qualifier in the names of some objects. It identifies an owning object. DataAtlas, not the user, adds it to the name, uses it, and maintains it.

access-name
The short, simple name of an object; the name by which it is best known outside the context of DataAtlas.

variation
An optional qualifier for the name of an object. It can be useful in distinguishing between similar objects that have different uses. For example, two shareable data elements could have the access name, ZIPCODE, but have different data characteristics for different applications. The variation qualifier could designate the application to which each belongs.

revision
An optional qualifier for the name of an object. It can be useful in distinguishing between objects that have to be available at the same time. For example, there could be two revisions of the same shareable data element named ZIPCODE: one representing a 9-digit zip code, one representing a 5-digit zip code, both co-existing within the same TeamConnection version. The revision qualifier could be used to differentiate them.

Some database objects have additional name qualifiers. The qualifiers precede the access name and indicate a hierarchy in which the object exists. For example, PAYROLL_TABLE, a table object, has the name `<>m78serv3:userid:servsamp::PAYROLL_TABLE<v:r>`. The `system` qualifier is `m78serv3`, the `creator` qualifier is `userid`, and the `database` qualifier is `servsamp`. Notice that qualifiers are separated from each other by a colon and from the access name by a double colon.

The only part of a name that is required is the access name, which you choose and maintain. The full length of the name, qualifiers included, can be up to 255 characters long, including special characters. No limit exists on any part of a name, except for variation and revision which each have a limit of 30 characters. When you choose an access name, variation, and revision, take into account the length of the other qualifiers. Since the characters < and > and : are used to separate the parts of a name, they cannot be used within the access name, variation or revision values. In addition, TeamConnection does not allow \ (backslash) or | (vertical bar) in part names. See “Appendix B. Qualifiers for Object Names” on page 141 for more information about using object name qualifiers.

Examples for Valid Object Names:

DB2 UDB table

<>m78serv3:userid:servsamp::PAYROLL_TABLE<:r1>

Oracle table space

<>m78serv2::::CUSTLST<:>

IMS DBD

<>BE3ORDER<V1:R1>

IMS PSB

<>PE3ORDER<:>

IMS PCB

<>PE3ORDER::PCB1<:>

Included source definition

<>ABC.CPY<:>

Data element alias

<ZIPCODE_9>zipcode<:>

Shareable data element

<>ZIPCODE_9<V1:R0>

Shareable data structure

<>EMPLOYEE_REC<v1:r1>

Chapter 3. Searching for Objects in the TeamConnection Database

To view an object in the TeamConnection database, you will first need to create a TeamConnection work area in the family and release in which your object exists. When you are using TeamConnection change control, any task you perform is determined by a defect or feature. The work area is created to collect all the new and changed parts associated with the defect or feature. See *TeamConnection User's Guide* for instructions on creating and using a TeamConnection work area.

After you have created a TeamConnection work area, you can access objects in TeamConnection using a DataAtlas workfolder. A workfolder is a place to keep all the objects you need to perform a certain task, such as extending a database design by adding or modifying an IMS DBD or a set of relational tables. After the objects are in a workfolder, you can open a notebook view of the object, generate database or HLL (high-level language) source statements, or delete the object from the TeamConnection database. In this chapter, you learn how to create a workfolder and use it to search for objects in the TeamConnection database.

Creating a Workfolder

To create a workfolder:

1. Double-click the **Workfolders** icon in the DataAtlas Main Folder window.

   Result: The Workfolders window opens.
2. Click **Folder**, then **Create workfolder**.

   Result: The Workfolder notebook opens.
3. On the **General** page, enter a workfolder name and the component that the workfolder is to be associated with. On the **TeamConnection** page, fill in the **Family, Release,** and **Work Area** fields. Click **OK**.

   Result: The workfolder is created in the TeamConnection database. When the store is complete, the details view of the new workfolder appears in the Workfolders window.

Searching for Objects from a Workfolder

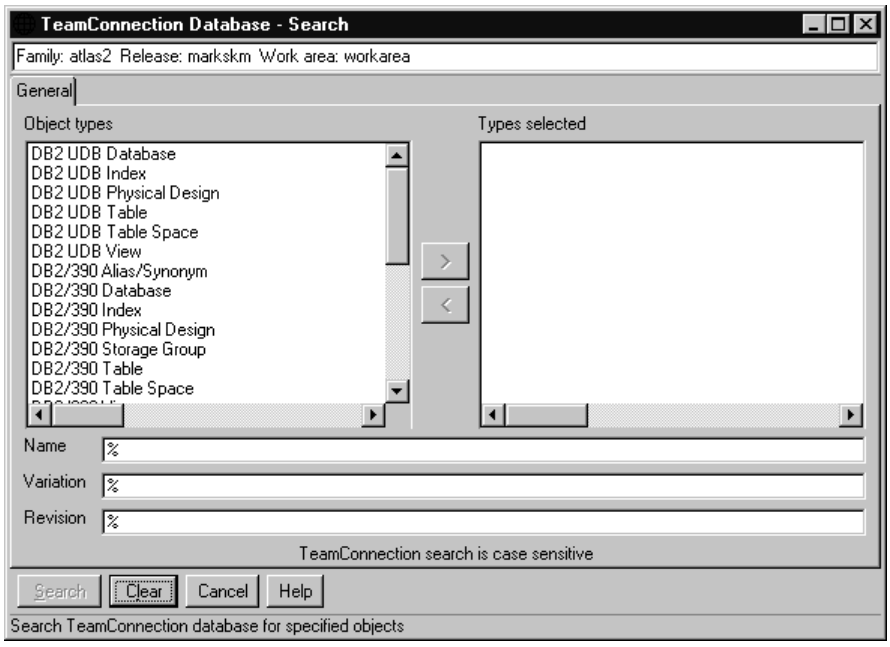
To search for an object from a workfolder:

1. In the Workfolders window, open a workfolder by double-clicking its icon. (If you're continuing from the last section, you will already have a workfolder open.)

   Result: The window for the selected workfolder opens.

2. Click **Workfolder**, then **Search TeamConnection database**.

   Result: The **TeamConnection Database - Search** notebook opens.



3. On the **General** page, double-click each object type in the **Object types** list that you want included in the search. You can also use the right and left arrows to move objects between **Object types** and **Types selected**.

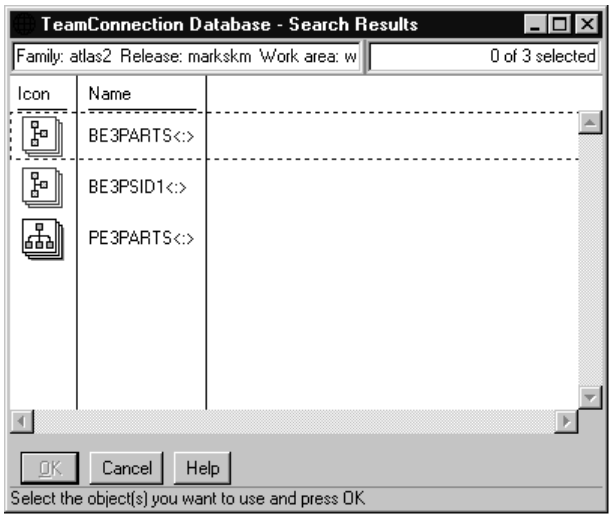
   Result: The types you selected are added to the **Types selected** list. You can deselect an object type in this list by double-clicking it.

4. You can qualify the search by specifying a **Name**, **Variation**, or **Revision**. Using **%** as a wildcard in each of these fields will find all instances of the object types you selected, within the family, release, and work area assigned to the workfolder.

5. Click **Search**.

   Result: The definitions matching the types you selected are displayed in the TeamConnection database - Search Results window.

The following screen shows the results of a search after populating some IMS DBDs from the sample files:



6. Select the objects you want to put into your workfolder. To select a list of items, select the first item, and press Shift and the left mouse button on the last item. To select separate items, select each by pressing Ctrl and the left mouse button. Click **OK**.

   Result: The objects are added to your workfolder.

Chapter 4. Creating and Deleting Objects

In DataAtlas, you can create these objects:

- DB2 UDB databases, tables, views, systems, indexes, and table spaces
- DB2/390 databases, tables, views, indexes, table spaces, storage groups, aliases, and synonyms
- Oracle tables, indexes, table spaces, and views
- IMS DBDs, PSBs, and PCBs
- COBOL and PL/I included source definitions
- Shareable data elements
- Shareable data structures
- Relational designs
- Physical designs
- Relational systems

Relational designs, physical designs, and relational systems are objects special to DataAtlas and don't correspond to objects in relational databases.

To create an object, you create its notebook, fill in the settings, and close the notebook to save your definition. You can press PF1 for help information on any field you are working on.

Creating Relational Database Objects

To create a DB2 or Oracle object, you must first create a relational system. A relational system represents an instance of a DB2 or Oracle catalog. Since the names of DB2 or Oracle objects must be unique within a catalog but tables with the same name may exist in different catalogs, the relational system name is used as part of the table name in the TeamConnection database.

To create a relational system:

1. In the DataAtlas Main Folder window, double-click the **Workfolders** icon.
2. Double-click the icon of the workfolder to which the system will belong.
   Result: The Workfolder window opens.
3. Click **Workfolder**, then **Create object**.
   Result: The Create Object window opens.
4. Click the plus sign to the left of **Relational Objects**.

Result: The list expands.

5. Click **Relational system**.

   Result: A notebook for a new relational system object opens.

6. On the **General** page, enter a name in the **Name** field.
7. On the **System** page, select **DB2/390, DB2 UDB**, or **Oracle** in the **Type** field. Select the correct version for the system in the **Version** field. Enter either a Creator ID in the **Creator** field, or a Schema ID in the **Schema** field, and click **Add**.
8. Click **OK**.

   Result: The System - Store window appears.

9. Click **Store**.

   Result: The newly created relational system is saved in the TeamConnection database and the window closes. The new relational system object appears in the workfolder.

In some cases, before a DB2 object can be created, a database must first be created. To create a DB2 database, select **Database** in the **Create Object** window, and select a release that agrees with the release you specified in creating the relational system object. The remaining steps are as follows:

1. In the Database notebook, click **Search** to fill in the **System** and **Component** fields.
2. Click **OK**.

   Result: The System — Store window appears.

3. Click **Store**.

   Result: The newly created DB2 database is saved in the TeamConnection database and the window closes. The new DB2 database object appears in the workfolder.

To create and define a relational table, table space, view, or index:

1. Click **Workfolder**, then **Create object**.

   Result: The Create Object window opens.

2. Click the plus sign to the left of **Relational objects**.

   Result: An expansion list is shown.

3. Click the plus sign to the left of either **table**, **table space**, **view**, or **index**.

   Result: An expansion list is shown.

4. Select either **DB2 Version 3.2**, **DB2 for MVS Version 4.1**, **DB2 for OS/390 Version 5.1**, **DB2 Common Server Version 2.1**, or **Oracle Version 7.3**.
5. Click **Create**.

   Result: The notebook opens.

6. On the **General** page, enter a name in the **Name** field. Click **Search** to fill in the required fields (**Schema**, **Creator**, and/or **Database**) for the object you are creating.
7. On the **Definition** page, complete the definition of the DB2 or Oracle object.
8. Click **OK**.

   Result: The System - Store window appears, where you can enter remarks.
9. Click **Store**.

   Result: The newly created relational object is saved in the TeamConnection database and the window closes. A representation of the object appears in the workfolder.

Creating IMS Objects

To create and define an IMS DBD, PSB, or PCB:

1. In the DataAtlas Main Folder window, double-click the **Workfolders** icon.
2. Double-click the icon of the workfolder to which the DBD, PSB, or PCB will belong.
3. Click **Workfolder**, then **Create object**.

   Result: The Create Object window opens.
4. Select **IMS DBD, IMS PCB,** or **IMS PSB** from the list.
5. Click **Create**.

   Result: The notebook opens.
6. For a DBD, select the access method from the list, and select **Set Access Method**. For a PCB, select the PCB Type from the list, and select **Set Type**.

   Result: Additional definition pages are added to the notebook.
7. On the **General** page, enter a name in the **Name** field. Fill in the fields on the other notebook pages.
8. Click **OK**.

   Results: The System - Store window appears, where you can enter remarks.
9. Click **Store**.

   Result: The newly created definition is saved in the TeamConnection database and the window closes. The new IMS object appears in the workfolder.

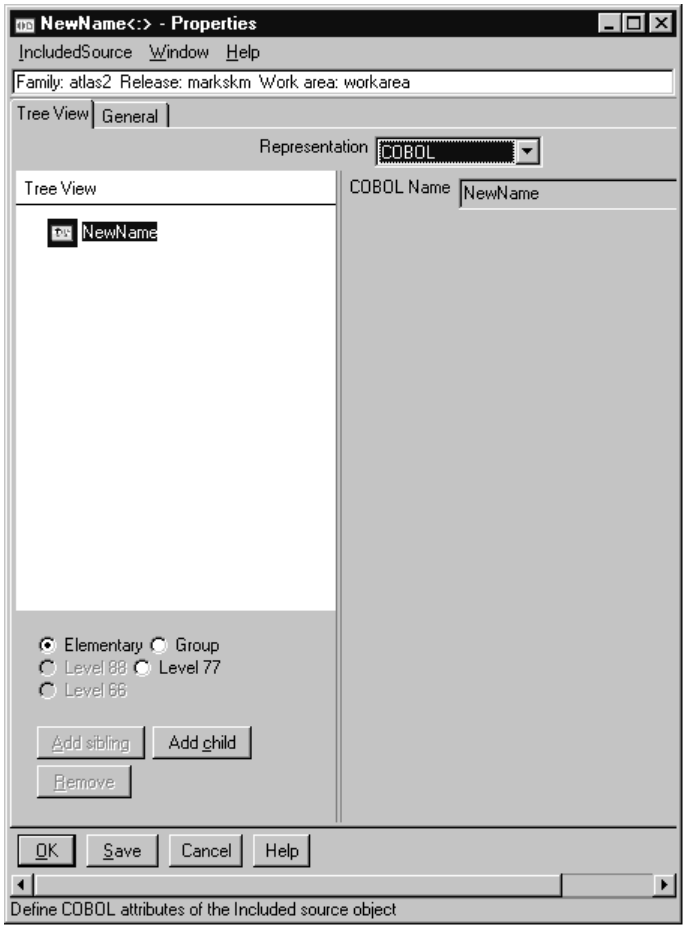
Creating Included Source Definitions

To create and define a COBOL or PL/I included source definition:

1. In the DataAtlas Main Folder window, double-click the **Workfolders** icon.
2. Double-click the icon of the workfolder to which the COBOL or PL/I object will belong.
3. Click **Workfolder**, then **Create**.

   Result: The Create Object window opens.
4. Select **Included Source Definition** from the list.
5. Click **Create**.

   Result: The notebook opens.



6. On the **General** page, enter a name in the **Name** field.
7. On the **Tree View** page, select either COBOL or PL/I from the **Representation** list.
8. Use the **Tree View** page to define the objects nested in an included source definition. The left-hand side of this page presents a tree view of the nested objects while the right-hand side of the page presents the details for each selected object. You can add new objects, modify existing objects, or remove objects from the structure.
9. To add a data item, select the type of item you want to add; then click either **Add Sibling** or **Add Child**, as appropriate. If the data item is based on an existing shareable data component, use the details information on the right-hand side of the page to specify the data component name as follows:
   a. If you choose to add an elemental item, select **Use existing shareable data element** to connect the item to an existing shareable data element.
   b. If you choose to add a group item, you must use an existing shareable data structure.
   c. Click **Search** to fill in the **Name** of the data component you want to share.

   To modify a selected data item, update the associated details information on the right-hand side of the page. If the data item uses an existing shareable data element or structure, then use **Open** to open and update the data component's notebook as necessary.

   To disassociate a selected data item from a shareable data element, deselect **Use existing shareable data element**. This does not remove the selected item, but makes it non-shareable instead. Note that structures contained in an included source definition must be shareable.

   To delete a data item from the data structure, select the item in the tree view; then click **Remove**.
10. Click **OK**.

    Result: The System - Store window appears, where you can enter remarks.
11. Click **Store**.

    Result: The newly created included source definition is saved in the TeamConnection database and the window closes. A new included source definition appears in the workfolder.

Creating Shareable Data Elements

A data element is the most elementary object used by IMS fields, DB2 columns, Oracle columns, and COBOL and PL/I elementary data items. DataAtlas Dictionary allows you to create *shareable data elements*, data elements that can be reused. An example of a data element that may be useful as a shareable data element is ZIPCODE. You can create a shareable data element of ZIPCODE and have other objects use it. Reusing a definition makes data more consistent and easier to administer.

To create a shareable data element:

1. In the DataAtlas Main Folder window, double-click the **Workfolders** icon.
2. Double-click the icon of the workfolder to which the data element will belong.
3. Click **Workfolder**, then **Create object**.

   Result: The Create Object window opens.
4. Select **Shareable Data Element** from the list.
5. Click **Create**.

   Result: The notebook opens. The **Representation** list on the **Definition** page determines the details used to describe the shareable data element.
6. Fill in the fields in the notebook.
7. Click **OK**.

   Result: The System - Store window appears, where you can enter remarks.
8. Click **Store**.

   Result: The newly created shareable data element is saved in the TeamConnection database and the window closes. A new shareable data element appears in the workfolder.

You can now associate the new data element with another object by opening the other object's notebook and filling in the appropriate field.

Creating Shareable Data Structures

A data structure is a collection of data elements. In the following steps, you'll learn how to create and define a shareable data structure. Making a data structure shareable means that it can be used repeatedly to define IMS segments or HLL data structures. Like the shareable data element, using shareable data structures makes data more consistent and easier to administer.

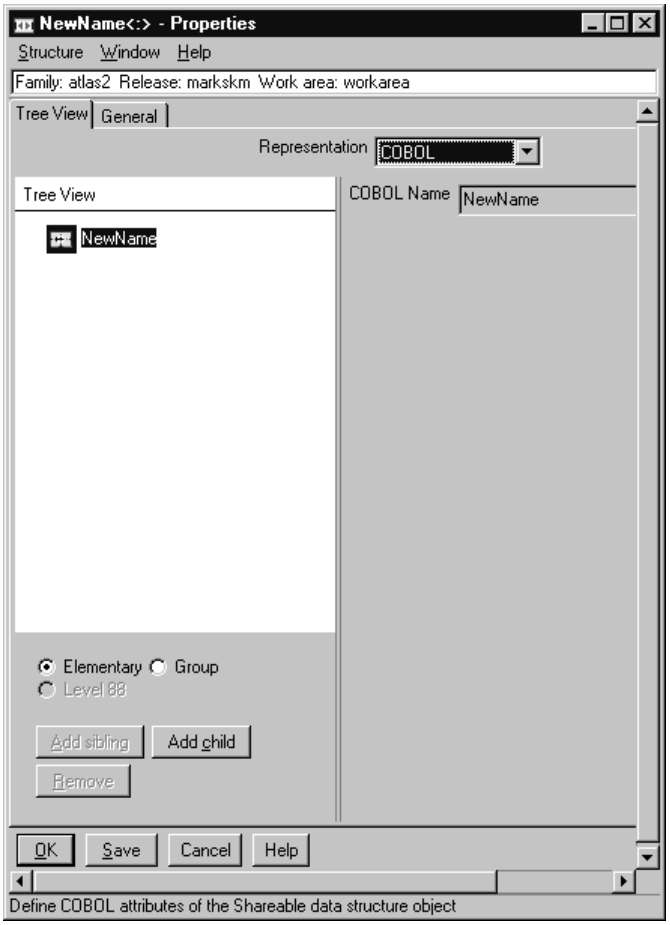
To create a shareable data structure:

1. In the DataAtlas Main Folder window, double-click the **Workfolders** icon.

2. Double-click the icon of the workfolder to which the data structure will belong.
3. Click **Workfolder**, then **Create object**.

   Result: The Create Object window opens.
4. Select **Shareable Data Structure** from the list.
5. Click **Create**.

   Result: The notebook opens. The **Representation** list on the **Tree View** page of the notebook determines the details used to describe the shareable data structure.



6. Fill in the fields in the notebook.

   Use the **Tree View** page to define the objects nested in a shareable data structure. The left-hand side of this page presents a tree view of the nested

objects while the right-hand side of the page presents the details for each selected object. You can add new objects, modify existing objects, or remove objects from the structure.

To add a data item, select the type of item you want to add; then click **Add Sibling** or **Add Child**, as appropriate. If the data item is based on an existing shareable data component, use the details information on the right-hand side of the page to specify the data component name as follows:

a. If you choose to add an elemental item, select **Use existing shareable data element** to connect the item to an existing shareable data element.
b. If you choose to add a group item, select **Use an existing shareable data structure** to connect the item to an existing shareable data structure.
c. Click **Search** to fill in the **Name** of the data component you want to share.

To modify a selected data item, update the associated details information on the right-hand side of the page. If the data item uses an existing shareable data element or structure, then use **Open** to open and update the data component's notebook as necessary.

To disassociate a selected data item from a shareable data component, deselect **Use existing shareable data element** or **Use existing shareable data structure**. This does not remove the selected item, but makes it non-shareable instead.

To delete a data item from the data structure, select the item in the tree view; then click **Remove**.

7. Click **OK**.

   Result: The System - Store window appears, where you can enter remarks.
8. Click **Store**.

   Results: The newly created data structure is saved in the TeamConnection database and the window closes. A new shareable data structure appears in the workfolder.

You can now associate the new data structure with another definition by opening the other definition's notebook and filling in the appropriate fields.

Deleting Objects

You can delete objects from a workfolder or from the TeamConnection database. If you delete an object from a workfolder, it still remains in the TeamConnection database. To delete an object from either location, it must first exist in a workfolder.

To delete an object from a workfolder:

1. Double-click a workfolder to open it.
2. Click the object you want to delete with the right mouse button.

   Result: A pop-up menu opens.
3. Click **Delete**.

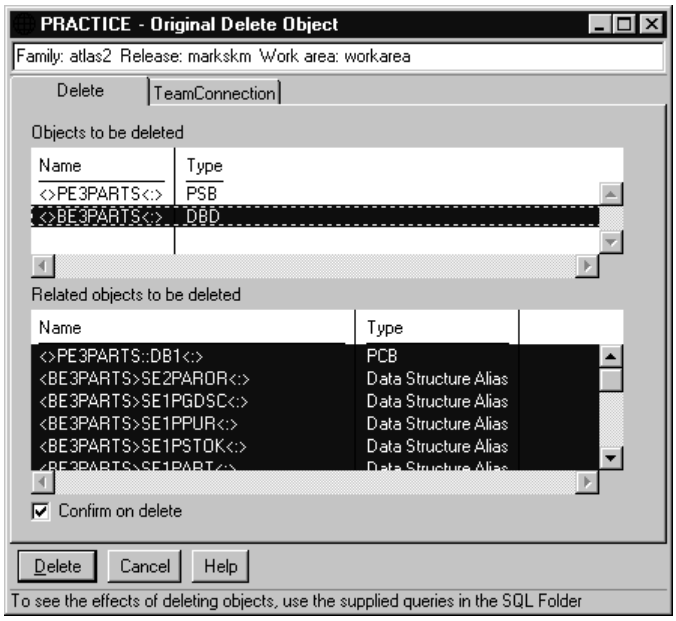
   Result: The object's icon disappears from the workfolder.

To delete an object from TeamConnection:

1. Double-click a workfolder to open it.
2. Click the objects you want to delete with the right mouse button.

   Result: A pop-up menu opens.
3. Click **Original**, then **Delete**.

   Result: The **Workfolder name - Original Delete Object** notebook opens.
4. The objects you selected for deletion appear in the **Objects to be deleted** list. Related objects are objects that are contained in or used by objects that you're deleting. These objects appear in the **Related objects to be deleted** list. In general, all related objects should be deleted when the selected object is deleted. You can use the query function to better understand where a particular object is used if you are unsure about whether the object should be deleted. See the topic "Supplied queries to use before deleting" in the online Help information.
5. You can deselect objects you do not want to delete from the **Related objects to be deleted** list.



6. Select the next object to be deleted from **Objects to be deleted**.

   Result: Other objects will appear in the **Related objects to be deleted** list. Deselect any objects you do not want to delete. Repeat this process until you have made a deletion decision about all the objects related to each of the entries in the **Objects to be deleted** list.

7. Select **Confirm on delete** if you want a report of the objects you've selected for deletion.

8. When you're done selecting objects for deletion, click **Delete**.

   Result: If you selected **Confirm on delete**, a list of the objects you selected appear in the Delete Confirmation window. If you want to delete all the objects, click **Delete**. If not, you must cancel the window and begin the object selection process over again.

   If you didn't select **Confirm on delete**, all selected objects are deleted from the TeamConnection database.

Chapter 5. Populating the TeamConnection Database

You can populate the TeamConnection database with DB2 data definitions that are defined in mainframe or workstation catalogs, from IMS source definitions and COBOL data definitions (from COBOL COPY files) that are stored in workstation files. DB2 data definitions are populated by specifying the name of a relational table. Any objects related to the relational table, such as indexes, views, databases, table spaces, and storage groups are populated automatically.

In this chapter, you'll learn how to populate a DB2 UDB table from the DA_CELD database, an IMS DBD, and a COBOL COPY file. The scenarios use sample files located in workstation directories created during the installation of DataAtlas. See "Appendix A. Sample Files Shipped with DataAtlas" on page 137 for a complete listing of the sample files shipped with DataAtlas.[1] When you finish the scenarios, you can try to populate your own data.

The basic scenario for populating from any of these sources is the same. First, select the appropriate source under **Populate from** in the Main Folder window. Then, from the source's pop-up menu, click **Populate** to see a list of candidates. Last, fill out the appropriate Populate notebook fields and click **Populate**.

Unless otherwise specified, the populate function creates a local data element for subcomponents such as a field within a DBD. You have the option of making that local data element a shareable data element, with the Reconcile function, as described in "Reconciling Data Definitions during Populate" on page 38 . During a COBOL Populate, a shareable data structure is automatically created for each 01-level data item structure. For other subcomponents, local data elements are created. Local data elements, shareable data elements, and shareable data structures are described in "Chapter 1. Introduction to DataAtlas Dictionary" on page 3.

1. Check with your administrator for the location of the sample files.

Populate Considerations When Using DataAtlas Modeler or Designer

Populate is an important step. It is usually done only once for each table in your system. After you have populated all the tables you want to manage with DataAtlas, there is no need to populate again. DataAtlas Designer has additional requirements that can be specified during populate.[2] The **Populate** notebook contains an entry field for **Relational Design** and **Physical Design**. When you specify values for either or both of these fields during populate, you are grouping the tables you selected to be used with DataAtlas Modeler and Designer. If you omit this step when populating, you must later update relational and physical design notebooks.

Relational and physical design objects are created using the **Relational Design** notebook from a workfolder. You must create these objects before you populate. See "Chapter 4. Creating and Deleting Objects" on page 19 for instructions on creating objects. The following section describes relational design and physical design objects:

Relational Design

This object is required by DataAtlas Modeler. Database design from a relational perspective focuses on table objects and the relationships between them. A relational design object in DataAtlas allows you to collect tables that you want to treat as a unit when using DataAtlas Modeler. You must specify the relational design name when you transform your tables from the relational definition used by DB2 to the entity-relationship used by DataAtlas Modeler. Since you may be populating many tables at once, it's important to identify the related tables and how they will be part of the same relational design.

Physical Design

Physical designs are a convenient way to organize your relational objects. For DB2/390, the physical design also allows you to access the DataAtlas Designer functions. Database design from a physical perspective focuses on a table's storage and access structures within the specific target database system. Typical tasks in a physical design are the creation of indexes to optimize the access path or the assignment of tables to appropriate table spaces. A physical design object in DataAtlas allows you to collect all the tables, indexes, table spaces, storage groups, and databases that you want to treat as a unit when using DataAtlas Designer.

2. See *DataAtlas Designer User's Guide* for complete information.

Since you may be populating many tables at once, it is important to identify related tables and how they should be part of the same physical design. In most cases, the tables you associate with a given relational design will also be associated with a given physical design during Populate. The additional physical objects that are populated with the table (indexes, views, databases, table spaces, and storage groups) also become related to this physical design object name.

Populating the TeamConnection Database from DB2 Catalogs

In the following scenario, you'll learn to populate the TeamConnection database with a sample table from the DA_CELD sample database.

Before you start, make sure you have:

1. Installed the DA_CELD sample database sent with the DataAtlas package
2. Started DB2 UDB (see "Chapter 2. Getting Started" on page 9 for more information).
3. Created a workfolder, DB2 system of the correct type, and a DB2 UDB database. A DB2 UDB database is necessary only when populating DB2 UDB tables. (See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for instructions on creating these).
4. If you will be using the DataAtlas Modeler or Designer, you must create relational and physical design objects using the Relational Design notebook from your workfolder. (See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for instructions on creating objects.)

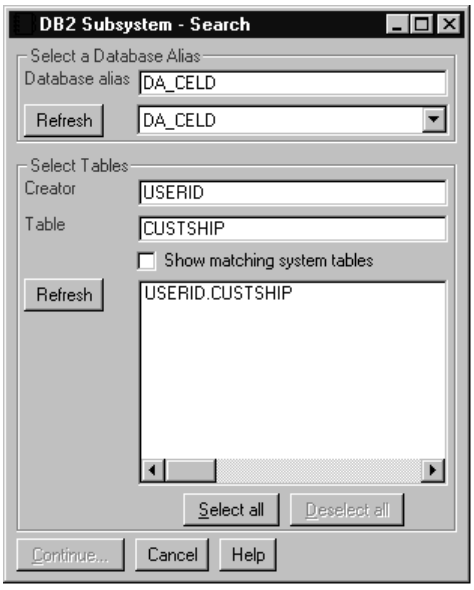
To populate the CUSTSHIP table:

1. In the DataAtlas Main Folder window, click the plus sign to the left of the **Populate from** folder.

   Result: The folder expands to show its contents.
2. Click the **DB2 Catalog** icon with the right mouse button.

   Result: The pop-up menu opens.
3. Click **Populate DB2**.

   Result: The DB2 Subsystem - Search window opens.



4. Type 'DA_CELD' in the **Database alias** field and click **Refresh**.

   Result: 'DA_CELD' appears in the list of database aliases next to the **Refresh** button.
5. Select 'DA_CELD' from the list.
6. Enter the user ID of the database creator in the **Creator** field.
7. Enter 'CUSTSHIP' in the **Table** field and click the adjacent **Refresh** button.

   Result: 'CUSTSHIP' appears in the list of tables next to the **Refresh** button.
8. Select 'CUSTSHIP' from the list.
9. Click **Continue**.

   Result: The **DB2 Subsystem - Populate** notebook opens, showing the selected table listed in the **Tables selected** list next to its new TeamConnection database name.
10. Click **Search** to fill in the **System**, **Relational Design**, and **Physical Design** fields.
11. Go to the **TeamConnection** page, and ensure the **Family**, **Release**, and **Work Area** fields are correct. The default values are those specified in the profile.
12. Click **Populate**.

    Result: The selected table and its related objects are stored in the TeamConnection database and a Populate report is produced.
13. Close the report and cancel out of both the **Populate** notebook and the DB2 Subsystem - Populate window.

14. It is recommended that you freeze your TeamConnection work area before ending this session. See *TeamConnection User's Guide* for information.

Using a Reconcile Mapping Table during DB2 Populate

During a DB2 Populate, the reconcile mapping table can be used to find existing shareable data components in the TeamConnection database that match the ones in the mapping table. Shareable data components that are listed in the mapping table that don't already exist in the TeamConnection database are created as a result of populating.

Populating IMS Definitions

In this section, you'll learn how to populate an IMS DBD. The process for populating PSBs is the same as for DBDs.

You can populate IMS data definitions only source code. IMS DBDs and PSBs in object code, typically members in a DBDLIB or PSBLIB, can be populated after you convert them to source code. You can do this with the product IMS System Utilities/Data Base Tools (DBT) Version 2. It contains several useful functions: DBD/PSB/ACB Compare, Mapper, and Reversal. The function you use to convert object code to source code is Reversal.

You must populate some IMS data definitions in a prescribed sequence. Populate all the DBDs used by a PSB before you populate the PSB. Populate physical DBDs before you populate the logical DBDs that depend on them.

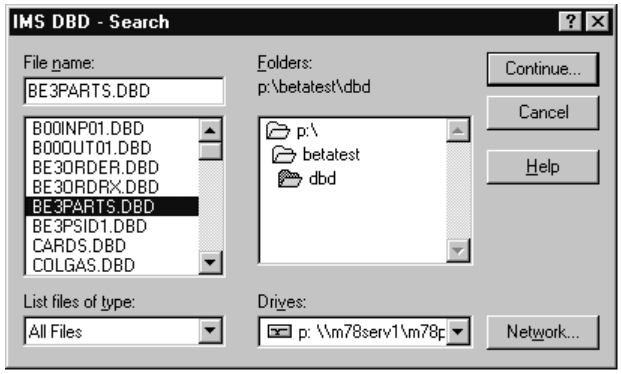
To populate the BE3PARTS DBD:

1. In the DataAtlas Main Folder window, click the plus sign to the left of the **Populate from** folder.

   Result: The folder expands to show its contents.
2. Click the **IMS Source** icon with the right mouse button.

   Result: The pop-up menu opens.
3. Click **Populate IMS DBD**.

   Result: The IMS DBD - Search window opens.



4. Select the drive where the sample files are located. [3]

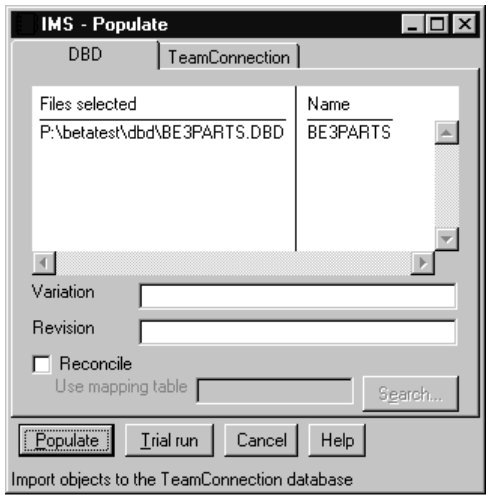
   Result: The **Directory** and **File** lists are updated to reflect the directories and files found on the drive you selected.

5. In the **Directory** list, double-click the subdirectory where the sample files are located.

   Result: The sample files are listed in the **File** list box.

6. Double-click BE3PARTS.DBD from the **File** list box. (You can also select multiple files, which will be populated sequentially.)

   Result: The **IMS - Populate** notebook opens, showing the selected file in the **Files selected** list next to its TeamConnection database name.



3. You can ignore this step and Step 5 if you've already identified a path to the sample files in your profile.

7. Go to the **TeamConnection** page and ensure the **Family**, **Release**, and **Work Area** fields are correct. The default values are those specified in the profile.
8. Click **Populate**.

   Result: The IMS DBD is populated into the TeamConnection database and a Populate report is produced.
9. Close the report.

   Result: You are back where you started with one important difference...you successfully added an IMS data definition to the TeamConnection database.

Using a Reconcile Mapping Table during IMS Populate

During an IMS DBD or PSB Populate, the reconcile mapping table can be used to find an existing DBD to relate to the new DBD. This information is specified in the mapping table as a special entry that has DSDBD specified as the source object type and the target object type. An example of an entry is shown in "Examples of Entries" on page 41.

The reconcile mapping table can also be used to process IMS segments and IMS fields. During an IMS Populate, a segment can be reconciled to a data structure. Fields within the segment can be reconciled to data items within the data structure. Segments and fields are processed prior to the normal reconciliation of data items to shareable data elements described later in this section.

Populating COBOL and PL/I Data Structures

In this section, you'll learn how to populate COBOL and PL/I data structures — that is, COBOL COPY files and PL/I include files. When populated, both exist in the TeamConnection database as included source definitions.

Don't try to populate COBOL or PL/I programs. DataAtlas may respond unpredictably, and may import incomplete data into the TeamConnection database.

Before you populate, try a COBOL or PL/I compilation from the session command line to be sure that the compiler is set up correctly.

The following steps show how to populate COBOL data structures. The same steps apply to populating PL/I data structures.

1. In the DataAtlas Main Folder window, click the plus sign to the left of the **Populate from** folder.

Result: The folder expands to show its contents.

2. Click the **COBOL Source** icon with the right mouse button.

   Result: The pop-up menu opens.

3. Click **Populate COBOL**.

   Result: The COBOL - Search window opens.

4. Select the drive where the sample files are located. [4]

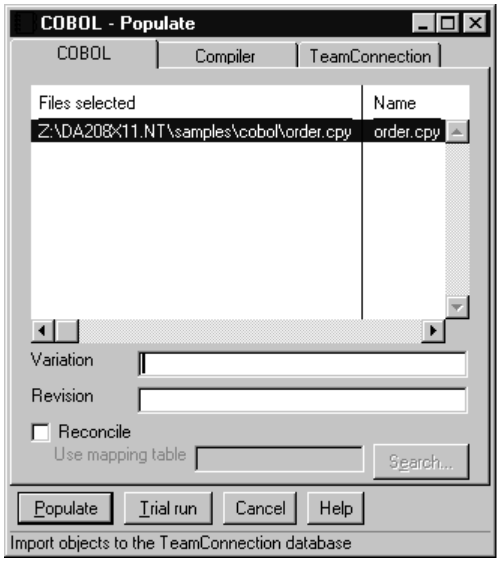
   Result: The **Directory** and **File** lists are updated to reflect the directories and files found on the drive you selected.

5. In the **Directory** list, double-click the subdirectory where the sample files are located.

   Result: The sample files are listed in the **File** list box.

6. Double-click ORDER.CPY from the **File** list box. (You can also select multiple files which will be populated sequentially.)

   Result: The COBOL - Populate notebook opens, showing the selected file in the **Files selected** list next to its repository name. The repository name can be changed by holding the ALT key while clicking on the name, overwriting it with the new name and clicking within the list.



7. Go to the **TeamConnection** page and ensure the **Family**, **Release**, and **Work Area** fields are correct. The default values are those specified in the profile.

4. You can ignore this step and Step 5 if you've already identified a path to the sample files in your profile.

The COBOL Populate function invokes the VisualAge for COBOL for OS/2 compiler with the ADATA, NOC, and NOEXIT options. This is the minimum set of options needed to produce the information required to populate a COBOL COPY file into the TeamConnection database. These options are shown on the Compiler page under **Compile options**.

You can specify additional options before clicking **Populate**, but they must be compatible with the options used by DataAtlas. For example, you may not specify NODATA as an additional option because the compiler would not generate SYSADATA.

8. Click **Populate**.

   Result: The COBOL included source definition is populated into the TeamConnection database and a Populate report is produced.

9. Close the report and cancel out of the **Populate** notebook and the COBOL - Populate window.

When you populate a COBOL COPY or PL/I include file, a shareable data structure is created automatically for each 01-level data item in the COBOL COPY or PL/I include file.

Using a Reconcile Mapping Table with COBOL or PL/I

When you populate a COBOL COPY or PL/I include file, the reconcile mapping table can be used to find existing shareable data structures or shareable data items to relate to the new data items in a COBOL COPY or PL/I include file.

When you populate a COBOL COPY or PL/I include file, data structures (data items containing subitems) and data items can be reconciled by checking the **Reconcile using mapping table** box and specifying a mapping table name in the **Mapping table** field. This allows you to create a new shareable data element or shareable data structure (or use an existing shareable data component) for each data item whose name, data type, and data length appear in the mapping table. See “Reconciling Data Definitions during Populate” on page 38 for more information.

Running a Trial Run

If you want to test a populate procedure without actually creating any TeamConnection database objects, click **Trial Run**. A Populate report is created for you to review and no objects are stored in the TeamConnection database. A trial run is useful when you are preparing a mapping table for reconciling objects.

Reconciling Data Definitions during Populate

With the Reconcile function, you use a mapping table to determine which of the local data components should be reconciled (mapped) to existing shareable data components, or which local data components should be made into shareable data components. A local data component can be a local data element or a local data structure. The mapping table contains a list of source object identifiers for local data components and a corresponding list of target object identifiers for shareable data components.

Check the **Reconcile using mapping table** box on the Populate notebooks if you want DataAtlas to use a shareable data component that might already exist in the TeamConnection database in place of the local data component that the default Populate action uses.

The **Reconcile using mapping table** check box applies only to objects that can be reconciled: IMS segments and fields, data items in language structures, or relational database column definitions. The box will be checked if you selected **Reconcile using mapping table** as the default in the profile for the type of object you are populating.

For each reconcilable object being populated, the DataAtlas Dictionary looks through a mapping table for a source name whose name, data type, and data length match the object. If a match is found, the corresponding shareable data component specified in the mapping table is used instead of the local data component being populated. If the target object does not exist, a new shareable data component is created and given the name specified in the mapping table. The new shareable data component is then used instead of the local data component being populated.

If a local data component being populated does not appear in the mapping table, it is not processed by Reconcile. This means that it is not shareable and is known only within the context in which it was populated.

Recommendation: If you plan to share data structures between IMS and COBOL or PL/I, populate and reconcile the COBOL and PL/I definitions before you populate and reconcile the IMS definitions. Populating in that order will preserve all the level of structure in the COBOL and PL/I definitions. Populating in the opposite order causes you to lose the COBOL and PL/I level of structure during reconcile processing.

You can customize the default mapping table supplied with DataAtlas, or you can create your own. See “Creating Shareable Data Components without Reconciling” on page 39 for instructions on setting up a custom mapping table.

To reconcile objects during Populate:

1. Check the **Reconcile using mapping table** box on any of the Populate notebooks.
2. If the mapping table name that was specified in your Profile notebook is acceptable, continue to the next step. Otherwise, specify the appropriate name and path for the mapping table.
3. Click **Populate**.

   Result: Shareable data components are created in the TeamConnection database where matches are found in the mapping table. A Reconcile section within a Populate report is produced. The Reconcile section contains the name of the mapping table used and a message describing how each shareable data component was processed. Possible results are:

   - A new shareable data component is created
   - The local data component was mapped to an existing shareable data component
   - The local data component was not found in the mapping table

Creating Shareable Data Components without Reconciling

When you populate a DB2 table, you can ask DataAtlas to treat every column as if it were specified in a mapping table, even though you're not using a mapping table. To do so, you check the **Use or create data elements for all columns** check box. If you use a mapping table and this check box, you're assured of creating shareable data elements for columns that aren't in the mapping table.

When you populate a COBOL COPY file or a PL/I include file, a shareable data structure is created automatically for each 01-level group data item.

Mapping Tables

The Reconcile function requires a mapping table, a table that shows associations between local data components and shareable data components. This section shows the structure of a mapping table and explains how to create one.

Structure of a Mapping Table

After a mapping table has been created and edited, its entries have this form:

SrcObjType SrcName SrcDataType SrcDataLength TargObjType |TargName|

where:

SrcObjType = the source object type: DSDataItem (for an IMS field or a language structure field), DSRColumnDefinition (for a relational column), DSDBD (for an IMS DBD), DSSegment (for an IMS segment), or DSField (for an IMS field within a segment) are the only classes of objects that can be reconciled.

SrcName = the source name. This is an access name (a name with no prefix, version, or revision qualifier).

SrcDataType = the source data type. A 2-digit number, which represents the data type for elementary data items (DSDataItem and DSRColumnDefintiion). For all other source object types, SrcDataType should be 00. SrcDataType appears in the Reconcile report produced by a Populate trial run that uses the **Reconcile using mapping table** option.

Table 1. Source Data Types

Type Code	Description	Applicable to SrcDataLength
00	Binary number	N
01	Packed decimal number	N
02	Zoned decimal number	N
03	Floating-point binary number	N
04	Floating-point decimal number	N
05	Bit string	Y
06	Single-byte character string	Y
07	Double-byte character string	Y
08	Mixed-character string	Y
09	Date	Y
10	Time	Y
11	Timestamp	Y
12	Index	N
13	Undefined	N

SrcDataLength = a 5-digit number that contains the precision for numeric data items and length for string data items. This value is shown in the Reconcile report. For other data types, SrcDataLength should be `00000`.

TargObjType= the target object type: DSDataElement (for DSDataItem or DSRColumnDefinition), DSDBD (for an IMS DBD), or DSDataStructure (for DSDataItem structure or IMS Segment).

TargName = the target name enclosed in vertical bars (|). The *TargName* can be a fully qualified name (access name with variation and revision) or simply an access name.

A minimal amount of error checking is performed for the mapping table. If any errors are detected, such as too few fields in an entry, no Reconcile processing is performed, the Populate is terminated, and no objects are stored.

Examples of Entries

Below are some examples of mapping table entries:

```
DSRColumnDefinition CUSTNO       06 00007 DSDataElement   |CustNbr<x:y>|
DSRColumnDefinition CUSTNAME     06 00040 DSDataElement   |CustNam<a:>|
DSRColumnDefinition REPNO        06 00005 DSDataElement   |SalesRep<:01>|
DSRColumnDefinition BAL90        01 00009 DSDataElement   |DA_Balance|
DSDataItem          KEYL         06 00004 DSDataElement   |KEYL|
DSDataItem          CITY         06 00060 DSDataElement   |City<:>|
DSDataItem          MONTH        06 00002 DSDataElement   |Month<:12>|
DSDataItem          CON_ADDR     00 00000 DSDataStructure |StdAddress|
DSDBD               COMMINDX     00 00000 DSDBD           |TAGENCY<:>|
DSDBD               HLPIMAN      00 00000 DSDBD           |HLPISEC<:>|
DSSegment        <DBDA>SEGA<:>   00 00000 DSDataStructure |Structure01|
DSField          <DBDA.SEGA>FLDA 00 00000 DSDataItem      |ABC|
```

KEYL, CITY, and MONTH are fields within IMS DBDs. The entries beginning with DSDBD are special entries that are used by IMS Populate and not by the Reconcile function. For a description of how they are used, see “Using a Reconcile Mapping Table during IMS Populate” on page 35.

Creating a Mapping Table from a Prototype

If you use a mapping table that contains no entries for some of the local data components, DataAtlas creates a *prototype mapping table* for them. The created table is called a prototype because the source names and the target names are the same; you’ll probably have to edit the prototype to get the target names you want.

To create a prototype mapping table that represents all the local data components you want to populate and reconcile, you have to be sure of two things:

- The populate operation that creates the prototype mapping table must be a **trial run**; that is, nothing must be actually populated.
- The mapping table used in the trial run must contain none of the local data components you eventually want to populate; otherwise, the prototype mapping table will have no entries for these components. For this purpose, you can use the sample mapping table that’s shipped with DataAtlas.

Here are the steps to follow:

1. In the **Populate** notebook, check the **Reconcile** box.

2. In the **Use mapping table** field, enter the path and file name of the sample mapping table:

   ```
   <DataAtlasInstallPath>\lang\en_us\samples\datatlas.map
   ```

   Where `<DataAtlasInstallPath>` is the root directory where DataAtlas is installed.

3. Click **Trial Run**.

   Result: No objects are stored in the TeamConnection database. The populate operation produces a report and a prototype mapping table. Their file names are the same except for the extensions: `.RPT` and `.MAP`, respectively.

4. Close the report window and cancel out of the **Populate** notebook.

5. Edit the prototype mapping table to reflect the target names you want for your shareable data components. (You'll find the prototype mapping table in the directory specified on the **General** page of your **Profile** notebook.)

Chapter 6. Updating Objects

You can update any object in three steps: double-click the object's icon, modify its notebook, and close the notebook to save your changes.

In this chapter, you'll see how to update relational database objects, an IMS DBD object, an IMS PSB object, an included source definition, and a shareable data object.

Updating Relational Database Objects

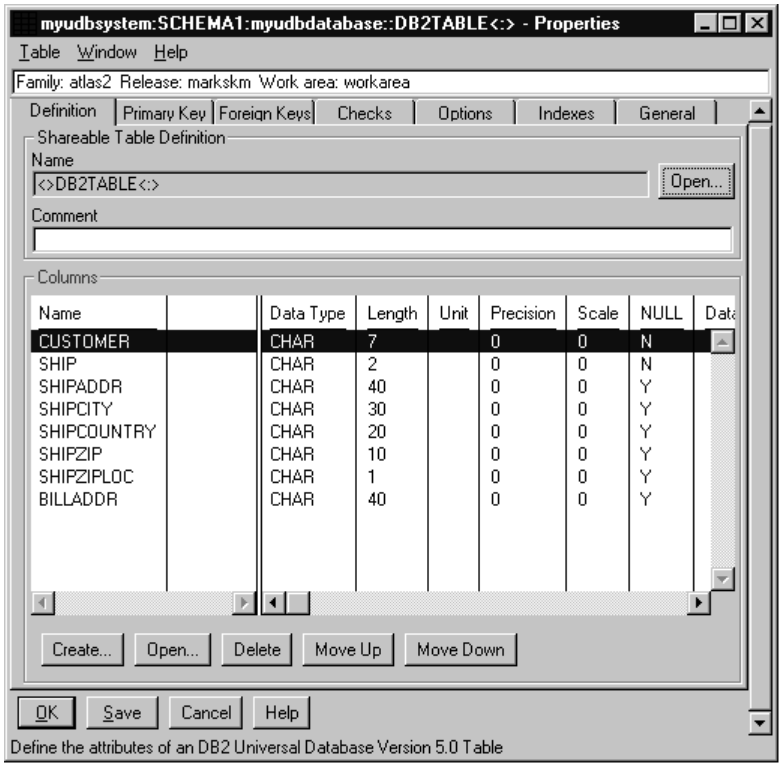
The following steps apply to a DB2 UDB table but similar steps are used to update any relational database object.

To update a DB2 UDB definition:

1. Double-click the workfolder that contains the object.
2. Double-click the icon for a DB2 UDB table.

   To locate objects use the Search function. See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for information on searching.

   Result: The properties notebook opens.



3. Update the fields in the notebook. Select a notebook tab to go from one page to another.

   To add a column to the table:

   a. Go to the **Definition** page.
   b. Click **Create**.
   c. In the displayed notebook, enter the column's name and other relevant information.

   If the column is based on a shareable data element, do this also:

   a. Enter a type in the **Type** field or click the arrow in the **Type** field and select from the expansion list. Leave the **Type** field blank to search for any sharable data element.
   b. Click **Search**.
   c. In the Search window, enter search criteria if none are already set, and click **Search**.
   d. When the search results appear, select the data element you want and click **Accept.**

To disassociate a selected shareable data element from a column, click **Remove**; then click **OK**. This will erase all data type information.

To delete a column from the table, select the column name in the column list; then click **Delete**.

4. Click **OK**.

   Result: The System - Store window appears, where you can enter remarks.

5. Click **Store**.

   Result: The notebook saves your updates and closes.

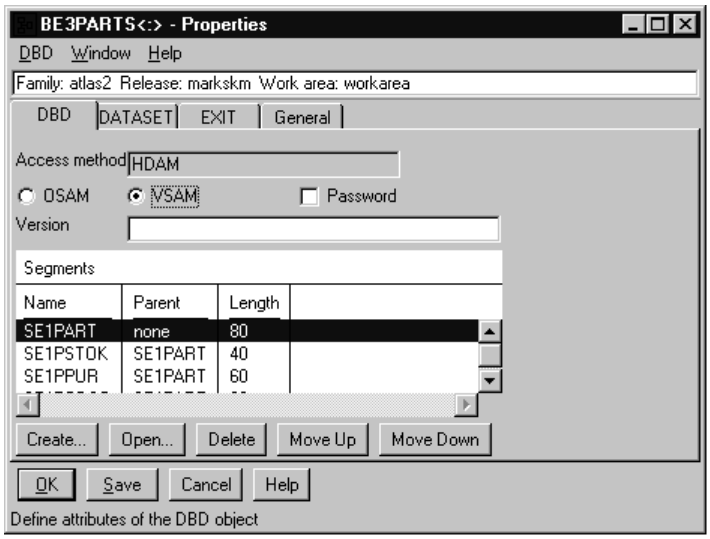
Updating IMS DBD Objects

To update an IMS DBD object:

1. Double-click a DBD icon in a workfolder.

   To locate definitions use the Search function. See “Chapter 3. Searching for Objects in the TeamConnection Database” on page 15 for information on searching.

   Result: The notebook appears. As an example, here is the notebook for one of the sample databases, BE3PARTS:



2. To add a segment to the database, click **Create**. A Segment notebook opens. You can later create new fields or delete fields from the segment. These actions change only the view of the underlying shareable data

structure. To change the shareable data structure, click **Open** to open the notebook for the shareable data structure.

3. To delete a segment, select the segment in the notebook you want to delete and click **Delete**.
4. After you're done updating the notebook, click **OK**.

   Result: The System - Store window appears, where you can enter remarks.
5. Click **Store**.

   Result: The notebook saves your updates and closes.

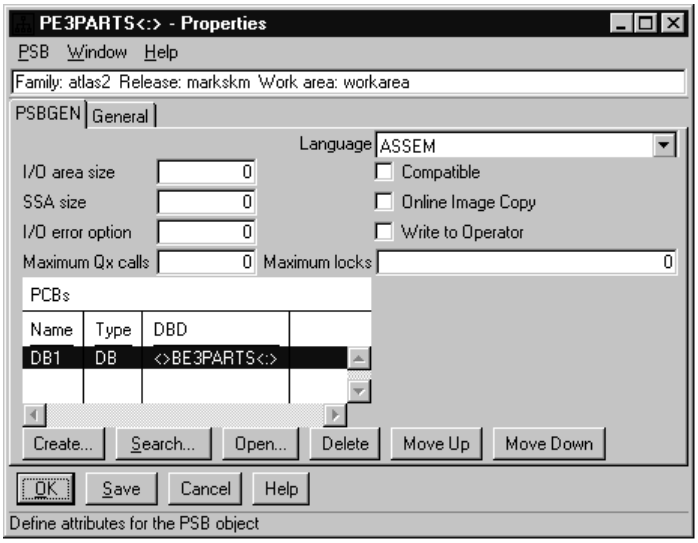
Updating IMS PSB Objects

To update an IMS PSB object:

1. Double-click a PSB icon in a workfolder.

   To locate definitions use the Search function. See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for information on searching.

   Result: The notebook opens. As an example, here is the notebook for a sample PSB, PE3PARTS:



2. To add a PCB to the PSB, click **Create**. A PCB notebook opens. Describe the new PCB by filling in the notebook fields, then click **OK**.
3. To delete a PCB, select the PCB in the notebook you want to delete and click **Delete**.

4. After you're done updating the notebook, click **OK**.

   Result: The System - Store window appears, where you can enter remarks.

5. Click **Store**.

   Result: The notebook saves your updates and closes.

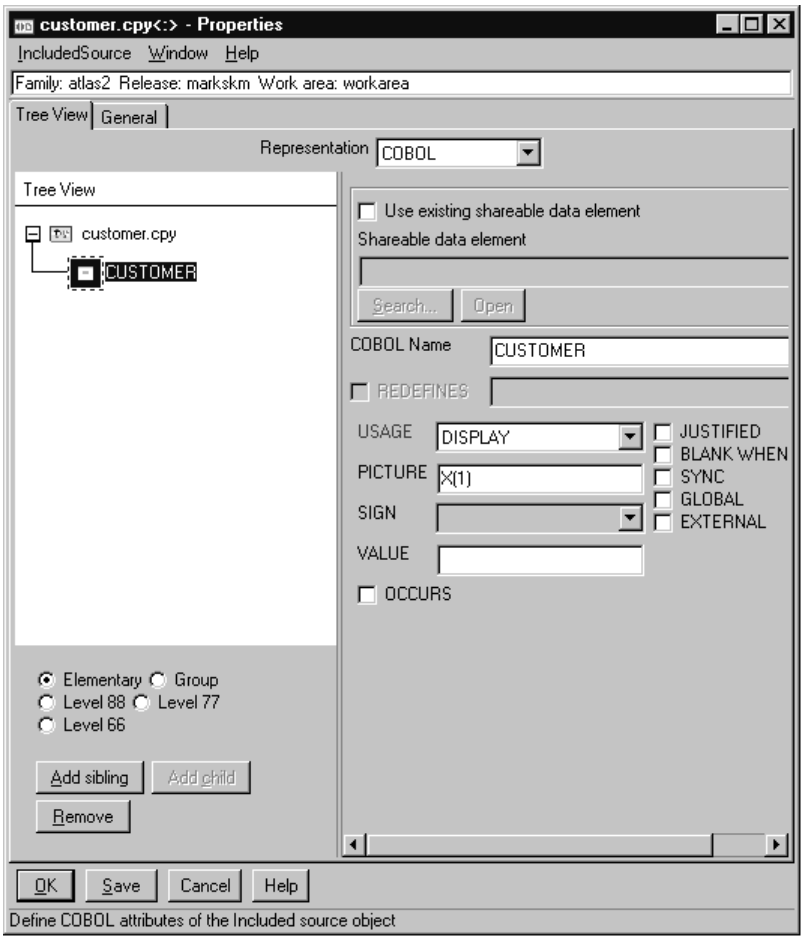
Updating COBOL or PL/I Included Source Definitions

To update an included source definition:

1. Double-click the icon for the included source definition.

   To locate these definitions use the Search or Query functions. See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for information on searching, and "Chapter 8. Running TeamConnection SQL Queries" on page 57 for information on running queries.

   Result: The notebook opens.



2. Update the fields in the notebook.

   Use the **Tree View** page to define the objects nested in an included source definition. The left-hand side of this page presents a tree view of the nested objects while the right-hand side of the page presents the controls for each selected object. You can add new objects, modify existing objects, or remove objects from the structure.

   To add a data item, select the type of object to add; then click **Add Sibling** or **Add Child**, as appropriate. If the data item is based on an existing shareable data component, use the control information on the right-hand side of the page to specify the data component name as follows:

   a. If you choose to add an elemental item, select **Use existing shareable data element** to connect the item to an existing shareable data element.

b. If you choose to add a group item, you must use an existing shareable data structure.

c. Click **Search** to fill in the **Name** of the data component you want to share.

To modify the selected data item, update the associated details information on the right-hand side of the page. If the item is associated with a shareable data component, then use **Open** to open and update its notebook.

To disassociate a shareable data component from the selected data item, deselect **Use existing shareable data structure** or **Use existing shareable data element**. This does not remove the selected item, but makes it nonshareable instead.

To delete a data item from the data structure, select the item in the tree view; then click **Remove**.

3. After you're done updating the notebook, click **OK**.

   Result: The System - Store window appears, where you can enter remarks.

4. Click **Store**.

   Result: The notebook saves your updates and closes.

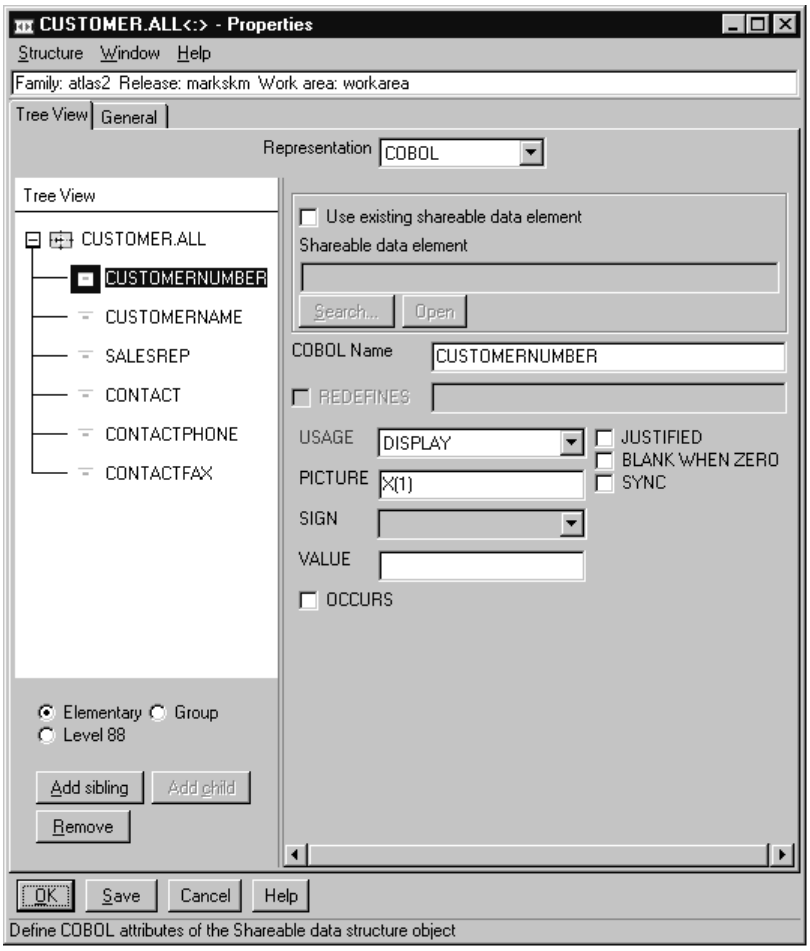
Updating Shareable Data Structures

To update a shareable data structure:

1. Double-click the icon for the shareable data structure.

   To locate these definitions use the Search or Query functions. See "Chapter 3. Searching for Objects in the TeamConnection Database" on page 15 for information on searching, and "Chapter 8. Running TeamConnection SQL Queries" on page 57 for information on running queries.

   Result: The notebook opens. The **Representation** list on the **Definition** page determines the **Details** used to describe the shareable data structure.



2. Update the fields in the notebook.

   Use the **Tree View** page to define the objects nested in a shareable data structure. The left-hand side of this page presents a tree view of the nested objects while the right-hand side of the page presents the controls for each selected object. You can add new objects, modify existing objects, or remove objects from the structure.

   To add a data item, select the type of item you want to add; then click either **Add Child** or **Add Sibling**, as appropriate. If the data item is based on an existing shareable data component, use the control information on the right-hand side of the page to specify the data component name as follows:

   a. If you choose to add an elemental item, select **Use existing shareable data element** to connect the item to an existing shareable data element.

b. If you choose to add a group item, select **Use existing shareable data structure** to connect the item to an existing shareable data structure.
c. Click **Search** to fill in the **Name** of the data component you want to share.

To modify a selected data item, update the associated details information on the right-hand side of the page. If the item is associated with a shareable data component, then use **Open** to open and update the component's notebook as necessary.

To disassociate a shareable data component from the selected data item, deselect **Use existing shareable data element** or **Use existing shareable data structure**. This does not remove the selected item, but makes it non-shareable instead.

To delete a data item from the data structure, select the representation of the element in the tree view; then click **Remove**.

3. After you're done updating the notebook, click **OK**.

   Result: The System - Store window appears, where you can enter remarks.
4. Click **Store**.

   Result: The notebook saves your updates and closes.

Chapter 7. Generating Definitions

When you generate a definition, you're doing almost the opposite of a populate: you are generating source statements from the objects that reside in the TeamConnection database. DataAtlas Dictionary retrieves the data from the TeamConnection database and writes the data as source statements. With DB2/390, DB2 UDB, and Oracle, the destination of the source statements must be a workstation file; if you want the source to execute on an OS/390 system, you must upload the file to the host. You also have the option of appending the new source statements to an existing file, or replacing an existing file altogether.

You can generate definitions for the following objects:

- DB2/390: DDL for tables, indexes, views, storage groups, databases, synonyms, and table spaces
- DB2 UDB: DDL for tables, views, and indexes
- Oracle: DDL for tables, indexes, views, and table spaces
- COBOL: COBOL COPY files
- PL/I: PL/I include files
- IMS: IMS macro statements for DBDs and PSBs
- Physical designs: DDL for all the objects in a physical design

If comments or labels have been specified, COMMENT ON or LABEL ON statements will be generated.

You can also generate description text, for use with DataGuide, for DB2 tables, Oracle tables, IMS DBDs, and IMS PSBs. To generate the text, request it from the **Generate to File** notebook.

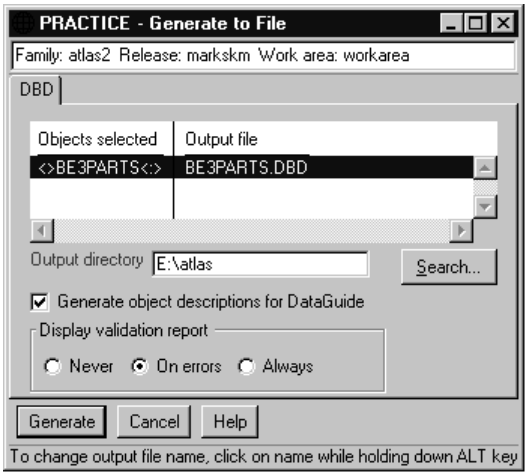
Generating Definitions from DataAtlas

To generate data definitions from the DataAtlas user interface:

1. Select from a workfolder one or more objects that reside in the TeamConnection database.
2. Click one of the selected objects with the right mouse button.

   Result: The pop-up menu opens.
3. Click **Generate to file**.

Result: The **Generate to File** notebook opens. Each page in the notebook represents a different type of selected object. The exception to this is the included source object which also has a compiler page.



4. For IMS, COBOL, PL/I, and physical design objects, specify a workstation drive and directory in the **Output directory** field. The output file can be changed by holding the ALT key down while clicking on the name, overwriting it with the new name and clicking on the list. For DB2 objects, specify the full path name for the file name.
5. If you use DataGuide, check the box called **Generate object descriptions for DataGuide**.

   Result: For IMS PSBs and IMS DBDs, descriptions are inserted in the generated output as comments. For DB2 UDB, DB2/390, and Oracle, the descriptions are put in a separate file with the name *creator.tablename*. DataGuide has special extractors for these description files.
6. Click **Generate**.

   Result: Definitions are generated to files in the locations you specified.
7. If the source you are generating is for execution on a 390 system, you need to upload the generated workstation file to the appropriate system.

HLL Cross-Generation

You can generate a COBOL COPY file from the PL/I representation of an included source definition. Likewise, you can generate a PL/I include file from the COBOL representation of an included source definition. However, you may first have to modify the included source definition to get successful results. Some of the syntax in each language is not supported by the other.

Generating Definitions with a Build Script

As an alternative to using the DataAtlas Dictionary user interface, you can use TeamConnection build scripts to generate data definitions. The TeamConnection builder uses a build script to invoke tools that transform one set of TeamConnection parts into another. You can create build scripts that invoke the DataAtlas program, DATATLAS.EXE, to generate DB2/390 DDL, DB2 UDB DDL, Oracle DDL, IMS DBD source, IMS PSB source, COBOL COPY files, and PL/I include files from data definitions stored in the TeamConnection database. For more information about using TeamConnection builders and build scripts, see *TeamConnection User's Guide*.

A sample build script is shipped with DataAtlas Dictionary. See "Appendix C. Using the DATATLAS.EXE Build Script" on page 145 for more information.

Using DataAtlas DDL on a DB2/400

DDL generated from DB2 UDB tables can be used on a DB2/400. Here are the steps to change, transfer, run, and view the DDL on a DB2/400.

Changing the Generated DDL on a DB2/400:

1. Create a collection for your user ID.
2. Remove any use of LONG.
3. Qualify VARCHAR fully (for example, VARCHAR(32739)).
4. When running RUNSQLSTM, make *SQL the naming convention.
5. Create a *LIB for your user ID.
6. Use Column 1 only for comments.

Transferring, Running, and Viewing DDL on a DB2/400:

Note: The following steps may not apply to all DB2/400 configurations.

1. Copy the DDL to a shared folder on the DB2/400.
2. Run CPYFRMPCD, on a DB2/400, to change the ASCII version of DDL to library text.
3. Run RUNSQLSTM, on a DB2/400, to create the database.
4. Use WRKSPLF, on a DB2/400, to view the listing.

Chapter 8. Running TeamConnection SQL Queries

This chapter shows you how to run TeamConnection SQL queries. It also includes a list of supplied TeamConnection SQL queries and instructions on how to write TeamConnection SQL queries. TeamConnection SQL is a superset of the SQL entry-level standard of 1992.

Running a Query

To run your own query (rather than one that DataAtlas has supplied), do the following:

1. In the Main Folder window, double-click the **SQL Query** folder.

   Result: The **SQL Query** folder opens.

2. Double-click the query you want to run. If the query is not among those in the **SQL Query** folder, click **Folder**, then **Create SQL Query**.

   Result: The **General** page of the **SQL Query** notebook opens.

3. If the query was not in the **SQL Query** folder, type in a unique file name or press the **Search** button to locate the file containing the SQL query.

4. Go to the **Query** page of the notebook and type in your SQL statement. If the query already exists, go to the **Query** page of the notebook to see the SQL statement.

5. Click **Run**.

   Result: The query runs and the **SQL Report** notebook is displayed with the query results.

6. To save your query to use or modify later, press **OK** to save the query and close the notebook. Press **Apply** to save the query and leave the notebook open.

   You can save the SQL query results in this report to view it later from DataAtlas. You can also export the results to an ASCII file, which can then be imported into a spreadsheet or reporting tools such as Lotus 123 or Lotus Approach.

Supplied Queries

DataAtlas supplies a set of queries to help you find what data is already defined in the TeamConnection database and what impact new data definitions may have on existing databases and application programs. You can also use TeamConnection SQL to write your own queries.

You can access the supplied TeamConnection SQL queries through the **SQL Query** folder in the **Main Folder**.

DataAtlas provides the following types of queries:

Impact analysis
The impact analysis asks what objects use the object in question. For example, you can query what objects use the shareable data element and shareable table definition objects so you can understand the impact of revising the shared data.

Glossary
The glossary queries show all the characteristics of an object or list of objects.

Relationship
The relationship queries provide a list of objects that are somehow related to the principal object. These queries are widely used for objects that are primarily containers of other objects, such as a Relational Design or DB2 System.

Running a Supplied Query

To run a supplied query:

1. Use the table in Table 2 to find the query type and name.
2. Double-click the **SQL Query** folder.

   Result: The **SQL Query** folder notebook opens, showing the supplied queries by query name.
3. Double-click the query you want to run.

   Result: The **General** page of the **SQL Query** notebook appears.
4. Go to the **Query** page of the notebook to view the query's description.
5. Click **Run** to run the query.

   Result: The supplied query is run against the sample data shipped with DataAtlas. You can modify the supplied queries to run against your own objects.

Table 2. DataAtlas Supplied Queries

Object Name	Query Type	Query Name	File Name
Data Element	Impact analysis	DataElementAliasUsage	EWSQDEAU.QRY
COBOL and PL/I Objects			

Table 2. DataAtlas Supplied Queries (continued)

Object Name	Query Type	Query Name	File Name
Included Source Definition	Glossary	IncludedSourceGlossary	EWSQISG.QRY
		IMS Objects	
IMS DBD	Impact analysis	DBDUsage	EWSQDU.QRY
	Glossary	DBDSegmentName	EWSQDSN.QRY
	Glossary	DBDAccessMethod	EWSQDAM.QRY
	Glossary	DBDFieldName	EWSQDFN.QRY
	Glossary	DBDIncomplete	EWSQDI.QRY
	Glossary	GSAMdbdData	EWSQID1G.QRY
	Glossary	MSDBdbdData	EWSQID2G.QRY
	Glossary	HSAMdbdData	EWSQID3G.QRY
	Glossary	DEDBdbdData	EWSQID4G.QRY
	Glossary	HDAMdbdData	EWSQID5G.QRY
	Glossary	HISAMdbdData	EWSQID6G.QRY
	Glossary	HIDAMdbdData	EWSQID7G.QRY
	Glossary	LOGICALdbdData	EWSQID8G.QRY
	Glossary	INDEXdbdData	EWSQID9G.QRY
	Glossary	SimpleSegmentData	EWSQIS1G.QRY
	Glossary	DEDBSegmentData	EWSQIS2G.QRY
	Glossary	ComplexSegmentData	EWSQIS3G.QRY
	Glossary	HISAMSegmentData	EWSQIS4G.QRY
	Glossary	LogicalSegmentData	EWSQIS5G.QRY
	Glossary	IMSfieldData	EWSQIFG.QRY
IMS PCB	Impact analysis	PCBUsage	EWSQPU.QRY
	Glossary	PCBGlossary	EWSQPG.QRY
		Relational Database Objects	
Relational Design	Relationship	RelationalDesignContents	EWSQRDC.QRY
DB2/390 Physical Design	Relationship	DB2390PhysicalDesignContents	EWSQDMPD.QRY
Relational Database system	Relationship	RelationalSystemContents	EWSQRSC.QRY
Owner ID	Relationship	RelationalOwnerContents	EWSQROC.QRY
DB2/390 Alias	Glossary	DB2390Alias	EWSQRA.QRY
DB2/390 Synonym	Glossary	DB2390Synonym	EWSQRS.QRY
DB2/390 Database	Relationship	DB2390DatabaseContents	EWSQDMDC.QRY
DB2/390 Table Space	Relationship	DB2390TablespaceContents	EWSQDMTC.QRY
	Glossary	DB2390TablespaceGlossary	EWSQDMTS.QRY

Table 2. DataAtlas Supplied Queries (continued)

Object Name	Query Type	Query Name	File Name
DB2/390 Table	Impact analysis	DB2390TableForeignKeys	EWSQDMFK.QRY
	Glossary	DB2390TableColumnName	EWSQDMCN.QRY
	Glossary	DB2390TableColumnData	EWSQDMCD.QRY
	Glossary	DB2390TableGlossary	EWSQDMTG.QRY
	Glossary	DB2390TableIncomplete	EWSQDMTI.QRY
DB2/390 Index	Glossary	DB2390IndexGlossary	EWSQDMIG.QRY
DB2/390 View	Glossary	DB2390ViewGlossary	EWSQDMVG.QRY
DB2/390 Buffer Pool	Impact analysis	DB2390BufferpoolUsage	EWSQDMBP.QRY
DB2/390 Storage group	Impact analysis	DB2390StorageGroupUsage	EWSQDMSG.QRY
	Glossary	DB2390StorageGroupGlossary	EWSQDMGG.QRY
MVS Volume	Impact analysis	OS390VolumeUsage	EWSQMVU.QRY
ICF Catalog	Impact analysis	OS390ICFCatalogUsage	EWSQMICU.QRY
DB2 UDB Database	Relationship	DB2udbDatabaseContents	EWSQD2DC.QRY
DB2 UDB Table Space	Glossary	DB2udbTablespaceGlossary	EWSQD2TS.QRY
DB2 UDB Table	Impact analysis	DB2udbTableForeignKeys	EWSQD2FK.QRY
	Glossary	DB2udbTableColumnName	EWSQD2CN.QRY
	Glossary	DB2udbTableColumnData	EWSQD2CD.QRY
	Glossary	DB2udbTableGlossary	EWSQD2TG.QRY
	Glossary	DB2udbTableIncomplete	EWSQD2TI.QRY
DB2 UDB Index	Glossary	DB2udbIndexGlossary	EWSQD2IG.QRY
DB2 UDB View	Glossary	DB2udbViewGlossary	EWSQD2VG.QRY
DB2 UDB Physical Design	Relationship	DB2udbPhysicalDesignContents	EWSQD2PD.QRY
Oracle Table Space	Glossary	OracleTablespaceGlossary	EWSQDOTS.QRY
Oracle Table	Impact analysis	OracleTableForeignKeys	EWSQORFK.QRY
	Glossary	OracleTableColumnName	EWSQORCN.QRY
	Glossary	OracleTableColumnData	EWSQORCD.QRY
	Glossary	OracleTableGlossary	EWSQORTG.QRY
	Glossary	OracleTableIncomplete	EWSQORTI.QRY
Oracle Index	Glossary	OracleIndexGlossary	EWSQORIG.QRY
Oracle View	Glossary	OracleViewGlossary	EWSQORVG.QRY
Oracle Physical Design	Relationship	OraclePhysicalDesignContents	EWSQORPD.QRY
		Shareable Objects	
Shareable Data Element	Impact analysis	DataElementUsage	EWSQDEU.QRY
	Glossary	DataElementDataCharacteristics	EWSQDEDC.QRY
	Glossary	DataElementAlias	EWSQDEA.QRY
	Glossary	DataElementGlossary	EWSQDEG.QRY

Table 2. DataAtlas Supplied Queries (continued)

Object Name	Query Type	Query Name	File Name
Shareable Data Structure	Impact analysis	DataStructureUsage	EWSQDSU.QRY
	Glossary	DataStructureGlossary	EWSQDSG.QRY
	Glossary	DataStructureAlias	EWSQDSA.QRY
	Impact analysis	DataStructureAliasUsage	EWSQDSAU.QRY
Shareable Table Definition	Impact analysis	TableDefinitionUsage	EWSQTDU.QRY
	Impact analysis	TableDefinitionForeignKeys	EWSQTDFK.QRY
	Glossary	TableDefinitionColumnName	EWSQTDCN.QRY
	Glossary	TableDefinitionColumnData	EWSQTDCD.QRY
	Glossary	TableDefinitionGlossary	EWSQTDG.QRY

Writing TeamConnection SQL

If you have written SQL queries, they probably refer to a *relational data model*—a model in which data is represented tabularly, in columns and rows. TeamConnection, in contrast, uses an *object data model*; data is represented as objects that have *attributes* and *relationships* among themselves.

Writing queries for an object data model offers you both a performance advantage and notational simplifications. However, you aren't required to use these simplifications. You can always write your queries as if they were against a relational data model, and the results will be the same.

Before you see what is different about TeamConnection SQL queries, consider the following comparison of the data models.

The Department-Employees Example

Imagine a company that keeps relational data in a department (DEPT) table and in an employee (EMP) table. The DEPT table has columns for department names (DNAME) and department numbers (DNUM). The EMP table has columns for employee names (ENAME), employee numbers (ENUM), department numbers (DEPTNUM), and phone numbers (PHONE). The relational tables look like this:

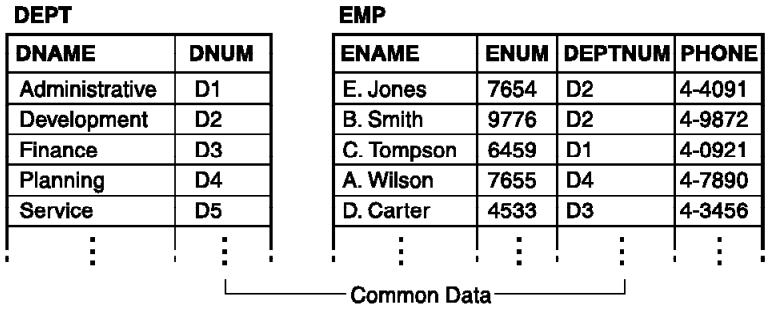


Figure 1. Relational Data Tables

Notice that the tables have columns with common data, DNUM and DEPTNUM. This data enables you to join the tables. You could, for example, write an SQL query that said, "Show me all the department names and, for each, the names of the employees in that department." The query would look like this:

```
select d.DNAME,e.ENAME from DEPT d,EMP e
   where d.DNUM = e.DEPTNUM;
```

In the object data model, DEPT would be the *class* of all the objects that contain department data, and EMP would be the class of all the objects that contain employee data. DEPT objects would have the attributes DNAME and DNUM, and relationships (call them ELINKS) to the EMP objects for the employees who work in a given department. EMP objects would have the attributes ENAME, ENUM, and PHONE. You can still visualize DEPT and EMP as tables and DNAME, DNUM, ELINKS, ENAME, ENUM, and PHONE as columns of these tables. The object tables (classes) would, in part, look like this:

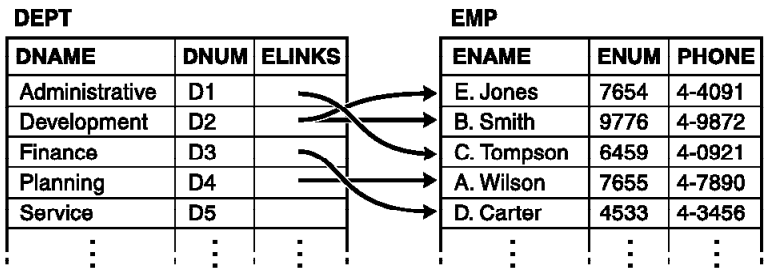


Figure 2. Object Data Tables

There are three important points to notice here:

- The rows of the DEPT and EMP tables correspond to objects within each class.
- The ELINKS column represents one-to-many relationships that objects in DEPT have with objects in EMP.
- These relationships, not duplicate data, are the means by which objects are linked. That's why there is no need for a DEPTNUM attribute in EMP.

Advantages of Querying an Object Data Model

TeamConnection SQL exploits the fact that objects are linked by their relationships. In gives you a new notation, the *dot-dot operator*, to express the linkage in queries. You can use this operator to write the above query ("Show me all the department names and, for each, the names of the employees in that department.") in a shorter form:

```
select d.DNAME,d.ELINKS..ENAME from DEPT d;
```

The dot-dot operator, written as two dots, shows the path from the relationships (ELINKS) in the source objects to the target attributes (ENAME) in the target objects. The values for the attributes DNAME and ENAME are returned in response to the query. Because of the linkage between objects, the response may be faster than the response to the conventional SQL query.

Using Qualifiers with Nested Collections

In the queries shown above, the `d` is called a *qualifier*; it says we are referring to the current instance of DEPT objects as `d`. In the object data model, an object can be related to many other objects. For example, a department can be related to multiple employees; the collection of employees is nested within the department object. You can name a new qualifier to refer to that nested collection.

For example, the query

```
select d.DNAME,d.ELINKS..ENAME from DEPT d;
```

can be written as

```
select d.DNAME,e.ENAME from DEPT d,(d.ELINKS)e;
```

In the second query, the qualifier `e` is bound to the collection `d.ELINKS` such that, whenever `d` is positioned on a department, `e` is positioned on the employees in that department. So writing `e.ENAME` is the same as writing `d.ELINKS..ENAME`. A restriction on using this notation is that the relationships being represented—ELINKS in this example—must be one-to-many relationships.

Of course, the simplicity of this example saves little notationally, but suppose you wanted to ask for more in the query, like department names and **all** the attributes in the EMP objects. Clearly, you could save time by simply writing

```
select d.DNAME,e.ENAME,e.ENUM,e.PHONE from DEPT d,(d.ELINKS)e;
```

Caution: Be careful in using multiple qualifiers to refer to the same collection. Each qualifier is bound to a different instance of objects. In a relational model, using the query `'select * from EMP e1, EMP e2'` names two qualifiers for the same table. In TeamConnection SQL, if you use the dot-dot operator to navigate a one-to-many relationship, an implicit qualifier is created. If you also explicitly name a qualifier for the nested collection, the qualifiers are bound to two instances of the collection, which is probably not what you want. In most cases you should name a qualifier for the nested collection and use that qualifier throughout the query.

Querying across Multiple Object Classes

You can use the dot-dot operator to query objects from multiple classes. Imagine, for example, that EMP objects have the relationships PROJLINKS to PROJ objects. These are objects that contain information about the projects employees are working on. PROJ objects might have attributes like PROJNAME, PROJLEADER, and PROJSCHEDULE.

Suppose you wanted to write a query that said, “Show me all the department names and, for each, the names of the employees in the department and the names of the projects they’re working on.” The query would look like this:

```
select d.DNAME,d.ELINKS..ENAME,d.ELINKS..PROJLINKS..PROJNAME
   from DEPT d;
```

Using qualifiers for simplification, you could write this equivalent query:

```
select d.DNAME,e.ENAME,p.PROJNAME from DEPT d,(d.ELINKS)e,
   (d.ELINKS..PROJLINKS)p;
```

Again, the usefulness of this shorthand depends on how extensive the query is.

Queries with an Outer Join

The queries in the examples above would not have returned any department names for departments without an employee. Nor would they have returned employee names for employees without a project. To see such cases returned in a query’s results, you use the keyword `OUTER` in the query. This keyword produces results similar to a *left outer join* in the relational data model. In the context of the previous query, the `OUTER` keyword would look like this:

```
select d.DNAME,d.ELINKS..ENAME,d.ELINKS..PROJLINKS..PROJNAME
   from OUTER DEPT d;
```

The results of this query would show a null under ENAME and PROJNAME for departments that have no employees. They would also show a null under PROJNAME for employees that have no projects.

TeamConnection's Views and Attributes

TeamConnection contains named views of all the object types that can be stored in its database. Each view contains named attributes for the object type it represents. For example, the view **DAQDB2390Table** is for DB2/390 table objects. The attributes `db2390Table` and `tableDescription` (two of many) refer to the name and description, respectively, of a DB2/390 table. "Appendix F. Views and Attributes of Object Types" on page 161 identifies all the views and attributes, and groups them by object type.

The supplied queries are built from these views and attributes. The SELECT clauses are made up of attributes appropriate to the query; the FROM clauses specify views or relationships to views. When you want to query the TeamConnection database, you can use these views and attributes to build your own query. But before you build one, consider starting with a supplied query and substituting attributes to produce the query report you want.

Part 2. Designer User's Guide

Chapter 9. Introduction to DataAtlas Designer

Database design is the link between your business data and optimum performance of your database system.

DataAtlas Designer is targeted to assist you, the database designer or the database administrator, in your complex and often time-consuming activities. DataAtlas Designer supports relational database design for the DB2 family of database systems, initially focusing on the MVS and OS/390 platforms.

DataAtlas Designer can help you after you create a relational design, a design that shows the relationships between tables and columns in terms of keys and constraints. You then use DataAtlas Designer to create a physical design to add physical information to the tables that will optimize storage of and access to your data. In creating a physical design, you take steps like the following:

- Estimate the data load and the work load for the application under design
- Define and associate additional resources (such as indexes, table spaces, buffer pools)
- Decide on physical adjacency of data (such as clustering indexes)
- Decide on placement of data (such as partitioning)

DataAtlas Designer provides notebooks that are structured for designing and maintaining database objects. In addition, DataAtlas Designer offers built-in design support through a powerful set of rules that compile and exploit DB2 internal knowledge. You get the following types of design support:

- Information on potential design problems
- Design proposals
- Validation of current designs

On request, proposed design steps are carried out automatically.

The process of database design can be viewed from two main perspectives:

- Forward engineering, the activity of creating databases
- Reverse engineering, the activity of improving existing databases

DataAtlas Designer supports both. In a forward engineering scenario, you use DataAtlas Designer to create a physical model from a relational model created with DataAtlas Modeler. With DataAtlas Designer, you add the physical information that is needed to achieve optimum performance for your applications on the target database system. The physical model is then

implemented in the database system, using the DataAtlas Dictionary function that generates the data definition language (DDL) needed for the database.

In a reverse engineering scenario, you use DataAtlas Dictionary to extract the data definition of an existing database from the database catalog and transform it into a physical model. Then you redesign this physical model with DataAtlas Designer. A redesign may be necessary, because you want to improve the performance, or because data load and work load conditions of the application changed. The improved physical model is then re-implemented in the database system.

Chapter 10. Database Design Concepts

This chapter contains a brief introduction to the main concepts encountered in database design for a relational database system, and how DataAtlas Designer deals with these concepts.

The Central Concept: The Table

The central concept in database design is the table object. The table object plays the central role from both perspectives, the relational perspective and the physical perspective.

Relational design focuses on tables and their interrelationships. Physical design focuses on how to store and access the tables in an optimum way.

DataAtlas Designer supports both types of design.

Relational Design

Database design from a relational perspective focuses on table objects and the relationships between them. This type of design is referred to as logical design, or, in the context of relational database systems, as relational design.

Typical tasks in relational design are the specification of primary keys, and the specification of foreign keys together with the referential constraints on them.

The following objects are involved when the focus of database design is on table objects and the relationship between them:

- Table
- Column
- Primary key
- Foreign key
- Unique key
- View

Physical Design

Database design from a physical perspective focuses on a table's storage and access structures within the specific target database system.

Typical tasks in physical design are the creation of indexes to optimize the access path for minimal costs, or the assignment of tables to appropriate table spaces.

The physical database objects depend on the specific type of database system. Physical database objects for DB2/390[5] comprise the following objects:

- Table
- Column
- Index
- Table space
- Database
- Buffer pool
- Storage group

Data Load and Work Load

For a specific database application, you must collect data load and work load information for each column of a table and for the table itself. This type of design information is crucial for adequate physical database design, whether you design your database objects directly or use DataAtlas Designer's design support. In all cases, the data load and work load data determines the adequate configuration of a table's storage and access structures.

Data load information is the basis for reasonable calculation of storage structures and related space allocations. Work load information, together with data load information, determines the configuration of the most efficient access path, which you achieve through the creation of indexes.

As data load information you need to provide, for example, the following input:

- Initial number of rows in a table
- Growth and delete rate
- Average column length for variable-length columns

5. This abbreviation refers to any of these DB2 enterprise data servers: DB2 Version 3, DB2 for MVS Version 4, and DB2 for OS/390 Version 5.

The work load is characterized by a set of operations and their relative frequency of application. The operations to investigate here are:

- Query
- Insert
- Update
- Delete
- Unload or Reload
- Reorg

Data load and work load information is essential for DataAtlas Designer's built-in design support. The rules that control the calculation of, for example, design proposals apply the current data load and work load data. If you don't enter sufficient data load information, DataAtlas Designer tells you what information is missing for the calculations it's making on your behalf. One of the most important data load values is the value for the initial number of rows. DataAtlas Designer uses it for many of the calculations it carries out.

In some cases, DataAtlas Designer assumes default values. DataAtlas Designer notifies you about the usage of default values so that, if needed, you can change them to actual values.

Storage

The assignment of appropriate storage resources is a major physical design task. For each table and index you create, you need to allocate auxiliary storage. DB2 provides the following concepts: Tables are assigned to table spaces and storage groups are used to allocate storage for table spaces and indexes. You can request design support from DataAtlas Designer for the optimum assignment.

If you do not want to use storage groups, you can also manage the allocation of storage yourself. In this case, you cannot request design support from DataAtlas Designer.

The major steps for data placement and space optimization are:

- Choose the appropriate type of table space for a table (simple, segmented, or partitioned)
- Calculate the space and free space requirements for the table space
- Use storage groups with appropriate device characteristics to split data according to its usage patterns (for example, indexes on fast devices)
- Identify wasted space situations and optimize by modifying the record length

DataAtlas Designer offers you design support on these design steps. Or, you can try to reach the same design objectives by adjusting values in object notebooks.

Access

The creation of indexes guarantees the most efficient access path to your data.

To create the optimum index configuration, you must have an overview of the work load of a given set of tables.

The goal is to minimize the total processing cost by selecting a set of appropriate indexes for each table. The total processing cost is the frequency-weighted sum of the expected costs for executing each statement, including access, tuple update, and index maintenance cost. An index configuration can consist of a set of clustered and non-clustered indexes.

For the creation of indexes, DataAtlas Designer offers you design support. Or, you can create the necessary indexes yourself by creating and initializing Index notebooks.

Knowledge of Database Design

Relational and physical database design is driven by a database designer's heuristic knowledge of database design.

DataAtlas Designer's notebooks take this heuristic knowledge into account and present a well-structured form for designing database objects.

In addition to common design functions, DataAtlas Designer offers design support based on rules. The rules represent collection of heuristic knowledge of DB2 database design. You can tailor the set of rules and customize individual rules to meet the needs of your application.

Chapter 11. Database Design with DataAtlas Designer

This chapter presents some characteristics of DataAtlas Designer and its approach to database design.

Design Modes

DataAtlas Designer provides two modes of design:

- Using notebooks
- Using built-in design support

For direct, manual design, DataAtlas Designer offers a set of notebooks. Design support can be requested for many design steps. It is based on a comprehensive set of heuristic rules, which you can tailor to your needs.

Both design modes are equally available throughout the database design process. From anywhere in a notebook, you can request design support. From a design advice report, you can open the notebook of an object that has been proposed, or for which potential design problems were detected.

You can apply both types of design to the creation as well as the optimization of databases.

Design Using Notebooks

DataAtlas Designer facilitates your database design by providing notebooks.

When you use notebooks for your database design, you are responsible for the data you enter into the provided fields. However, even though you may not make use of the design proposals DataAtlas Designer offers, you can still use the information and validation functions of DataAtlas Designer to check and verify the data you entered.

Design Using Design Support

DataAtlas Designer's most important feature is its function of design support. Design support is based on a collection of heuristic rules on database design. These heuristic rules take the provided data load and work load information as input for rendering reasonable design advice. For many design items, DataAtlas Designer can provide advice on the right step to take.

The type of support is threefold (the DataAtlas Designer function is given in brackets):

- Check completeness (Inform)
- Check correctness (Validate)
- Suggest improvements (Propose)

With each function, you can choose from a list of design items for which DataAtlas Designer can offer support. The advice is given in the form of a report that lists the design steps to consider. When design proposals are requested, you can accept a proposed design step, and DataAtlas Designer can carry out this step for you.

DataAtlas Designer gives information on potential design problems for all significant physical design objects:

- Storage group
- Database
- Table space
- Index
- Table
- Column

Design proposals can be requested from DataAtlas Designer for the following set of objects:

- Storage group
- Table space
- Index
- Table

Validation of current designs is available for the same set of objects.

Tailoring the Rule Set

Each design advice action is connected to a set of heuristic rules. You can access this set of rules and tailor it to your needs:

- Activate rules
- Deactivate rules
- Modify rules

Rules can be activated or deactivated. If all rules for an action are deactivated, the design step itself is deactivated.

Modifying the Rules

Some rules contain threshold values. They are enclosed in brackets (< >). These values can be modified as needed.

Other values that can be modified are DataAtlas Designer default values. DataAtlas Designer accesses data load and work load values when it generates a design proposal. If no information is available, or if it is incomplete, DataAtlas Designer uses predefined default values with its rules. DataAtlas Designer default values are listed after the rule. You can change the listed value, if needed.

Design Areas

With DataAtlas Designer, you can choose to conduct your database design on either of the following:

- Single database objects
- A set of database objects

Designing Single Objects

Single database objects are the basic area of design. For each type of object, a notebook is available that presents the needed items of design in a well-structured manner.

Notebooks are available for the following database objects:

- Table
- Column
- Index
- Table space
- Database
- Storage group
- View
- Alias/Synonym

No notebooks are provided for buffer pools and volumes. Both are important storage facilities to which other database objects are assigned, but they do not need to be designed the same way as the other database objects. The design information for volumes is connected to the storage group notebook. For buffer pools, no separate design information is entered.

Designing a Set of Objects

Predominantly, the tasks that apply to the entire set of objects cover the configuration of your design environment, such as setting your profile or setting specific defaults (see section “Configuring the Design Environment” on page 83).

When you use design support, both the Inform and the Validate function can operate on a set of objects. However, the Propose function is available only for single objects.

Design Information

When you begin your database design, you collect the estimated data load and work load for your database application. You collect the data load and work load for each column of a table and for the tables themselves. This type of design information determines the adequate configuration of the storage of and the access to the tables of your database application. DataAtlas Designer’s design support function takes the data load and work load values as input for the calculation of adequate design proposals. Refer to section “Data Load and Work Load” on page 72 for a description of the required data and work load values.

After you implemented the physical model of your database application into your target database system, DB2 keeps statistic records of your application in the DB2 catalog. DataAtlas Designer allows you to extract the most important DB2 statistical values from the DB2 catalog. You extract these values from the following database objects:

- Table object
- Table space object

From the table object, you extract the statistical values for the table itself, for all of its columns, and for all of the indexes you created for the table. From the table space object, you extract the statistical values for that table space object and for the storage group it is assigned to. You find the statistical values for the database objects on the DB2 Actuals page of the corresponding notebook. (See the online help for a description of the statistical values on the DB2 Actuals pages.) As soon as the table or the table space object has been implemented, you can extract the statistical values from the DB2 catalog, using the Extract Statistics function on the DB2 Actuals page of their notebooks.

The DB2 actual values are helpful design information for the optimization of database objects. From time to time, you extract the statistical values from the DB2 catalog, you compare these actual values with your initial data load and work load estimations, you refresh the estimates with actual values, and you enter new data load and work load estimations. This way, you not only get information on the current status of design, but also on the progress of your design. Based on the new data load and work load data, you can again request DataAtlas Designer to calculate the optimum design steps for you.

Design Reports

The design advice reports show you the results DataAtlas Designer generated for your selected set of design steps and actions. The structure of the report depends on the requested design function (Inform, Validate, or Propose) and the type of object being designed.

Each line represents an object, such as an index, a table space, or a proposed attribute value. The reported objects are the ones that DataAtlas Designer proposes to create or modify. You can accept single proposals or the complete list of proposals. DataAtlas Designer then creates new objects (for example, a new index) or changes existing objects. Alternatively, you can reject proposals, and no action is carried out by DataAtlas Designer.

If design information for an object is missing, DataAtlas Designer informs you about it in the report. You can directly open the notebook of the corresponding object from the report, and you can correct the necessary information.

In a validation report, each line represents an object in error. In an information report, each line represents an object together with the advice given on its potential design problems. Again, you can switch to the objects to correct them.

You can get more details for each report item. When requested, you see the detailed rule on which the calculation for the selected design step is based.

Design Scenarios

DataAtlas Designer allows for the following types of database design:

- Creating a new database
- Optimizing an existing database

Creating a New Database

After you create a relational model with a modeling tool like DataAtlas Modeler, specify entities and relationships, map them onto table definitions and tables, and normalized the tables, you can add the necessary physical information to them as follows:

- Enter necessary data into notebooks.
- Collect data load and work load information.
- Design the database objects:
  - Use notebooks.
  - Request design proposals.
- Get information on potential design problems.
- Validate the physical design.
- Implement the design by using DataAtlas Dictionary to generate DDL.

Optimizing an Existing Database

A need for optimization of database design objects is given when data load or work load data of an application has changed. There are two different ways to optimize your database:

- Optimizing an entire database design
- Optimizing single database objects

Optimizing an entire database design involves the following steps:

- Getting DB2 catalog information, using the Populate function
- Defining work environment
- Entering current data load and work load information
- Updating existing physical database objects
- Requesting proposals for optimum design
- Validating the current design
- Implementing the changes by generating the DDL

Optimizing single database objects involves the following steps:

- Selecting database objects to optimize
- Extracting DB2 actual values for these objects, using DataAtlas Designer's Extract Statistics function
- Reviewing estimated and actual data load information:
  - Refreshing the estimates with actual data load values
  - Entering new estimates for data load
- Updating the selected objects and their related database objects:

  - Entering necessary data into notebooks
  - Requesting design proposals
- Validating the revised design
- Implementing the changes by generating the DDL

Chapter 12. Using DataAtlas Designer

This chapter shows you how to use DataAtlas Designer for the design of a complete physical design or the design of single database objects. For each database design object, you find a section with a description of all the relevant design aspects of that object.

You get an overview of the design modes DataAtlas Designer provides: the use of notebooks, and the use of the design support. And you find instructions on how to configure your design environment.

Configuring the Design Environment

At the beginning of a comprehensive database design, you need to configure your system environment. Usually, you carry out these configuration tasks only once, and the values you set are valid throughout the entire physical design.

Configuring your global design environment consists of the following tasks:

- Setting default values
- Specifying the system environment data

You can set default values for the following items:

- System
- Database
- Creator
- Physical design

The following values are required as system environment information:

- Database name
- Storage group
- Buffer pools
- Storage management system
- Close option
- Mixed data
- Prefix and suffix for name generation:
  - Index suffix
  - Index prefix

– Table space prefix
– Foreign key prefix

You specify the system environment data in the Profile notebook. For a description of the Profile notebook, refer to “The Profile Notebook” on page 11
.

Requesting Design Support from a Physical Design

You can request design support from a physical design. It contains all the database objects belonging to a specific database application. In a physical design, you get an overview of the involved objects; you can then select the specific objects you want to design.

To request design support from a physical design, do the following:

1. Open a physical design in one of the following ways:
   - From a workfolder:
     a. Open a workfolder that contains a DB2/390 physical design.
     b. Double-click the physical design.
   - From a Relational Design notebook:
     a. Open a Relational Design notebook and go to the DB2/390 page.
     b. Select a physical design.
     c. Click **Open**.

   In the Physical Design notebook, you’ll see a separate page for each of the object types that can belong to a physical design. On each page, you’ll find a list of the objects in the physical design that correspond to the page tab.

   Figure 3 on page 85 shows an example of available table objects on the Table page of a Physical Design notebook.

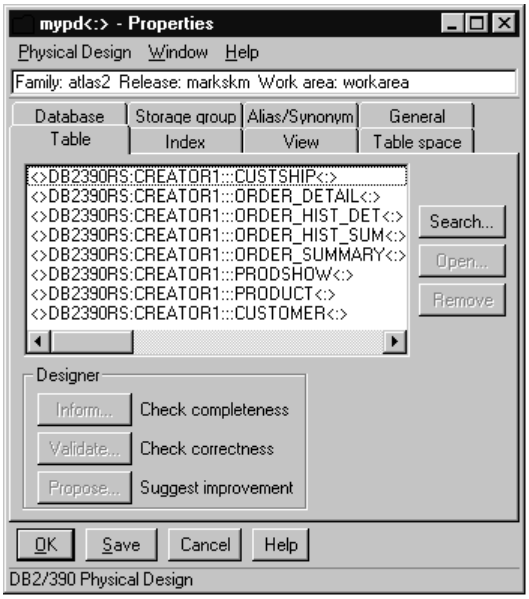


Figure 3. Table Objects in a Physical Design Notebook

2. Select the objects you want to design.
3. Click the push button that represents the design support you want:

   Inform, to check the objects' completeness

   Validate, to check their correctness

   Propose, to see suggestions for improvements

Requesting Design Support from a Workfolder

To request design support from a workfolder:

1. Open the workfolder and select the DB2/390 objects you want to design.
2. Click **Selected** on the menu bar, and then **Designer**.
3. Click a design function on the submenu.

Figure 4 on page 86 shows what a workfolder looks like when you click **Designer** on the menu bar.

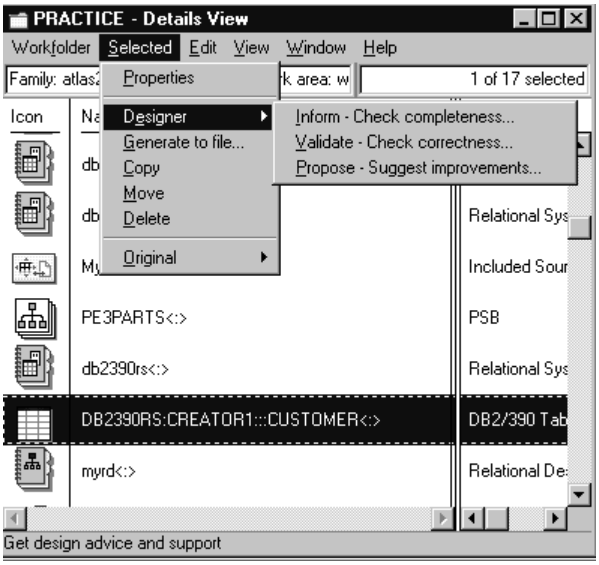


Figure 4. Design Functions on the Workfolder's Menu Bar

You can also request design support by opening an object's notebook from a workfolder and then invoking DataAtlas Designer from the notebook. "Requesting Design Support from a Notebook" on page 88 tells you where to look in a notebook for design support.

Designing Database Objects Using Notebooks

From an object's notebook, you can affect its design in three ways. You can design the object by tuning its definition in the notebook, you can request design support from the notebook, or you can do both.

Designing by Tuning an Object's Definition

At a high level, this is how to design an object by tuning its definition in a notebook:

1. Open a workfolder that contains the object you want to design.
2. Double-click its icon to open its notebook.
3. On the General page, fill in any blank fields that are required. (They have dark-pink labels.)

4. On the other notebook pages, define or redefine the object using the entry fields and options there. Figure 5 shows column data on the Definition page of a Table notebook.

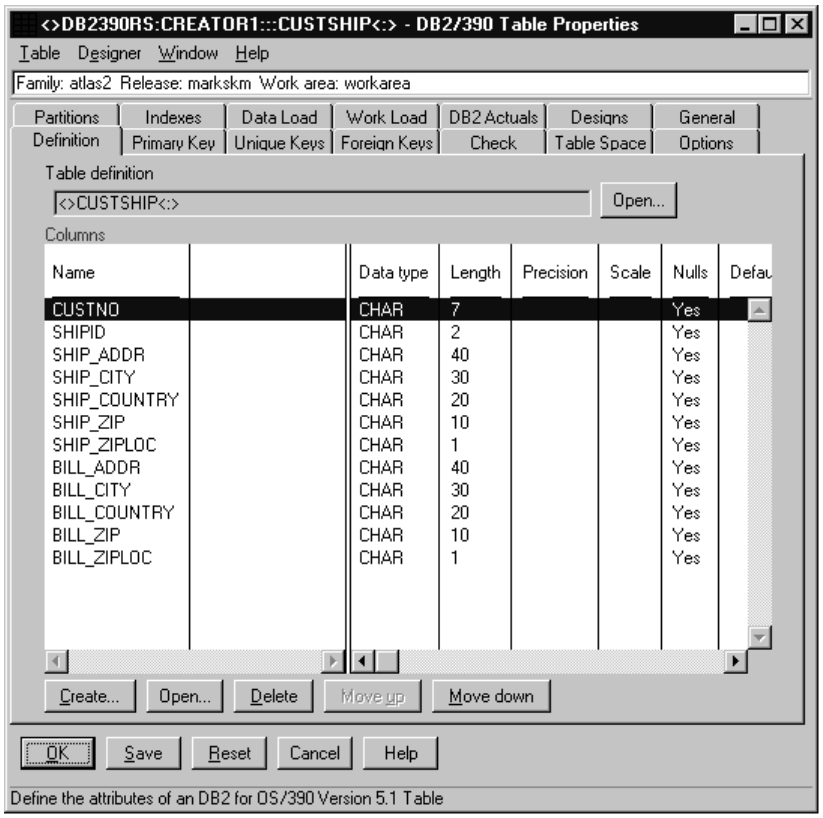


Figure 5. Table Notebook

5. Click **OK** when you're done.
6. Reopen the notebook from the workfolder when you want to tune the object's design further.

For a description of the fields in an object's notebook, click **Help** on the notebook.

For information on creating a new database object, refer to "Chapter 4. Creating and Deleting Objects" on page 19.

Requesting Design Support from a Notebook

To request design support from an object's notebook:

1. Click **Designer** on the menu bar.

   The pull-down menu shows the available design functions.
2. Click the design function you want.

The notebooks for columns, tables, indexes, storage groups, and table spaces have a Designs page with push buttons for the design functions available for these objects.

Selecting Design Actions

When you request design support, DataAtlas Designer opens a window showing the actions available for the design function you chose. Figure 6 shows the actions available when you click **Propose** for a table object.

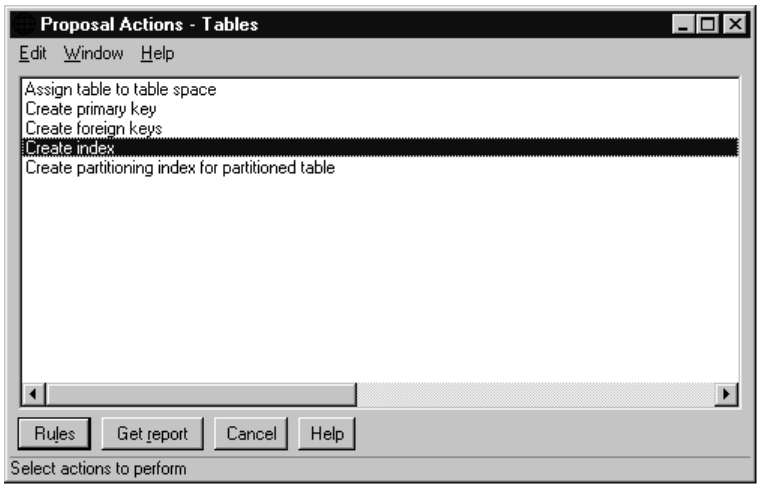


Figure 6. Available Proposal Actions for a Table Object

Each action consists of a number of action items. You determine which ones to select, and which of these to modify to meet the needs of your application.

To tailor a set of actions:

1. Select one or more.
2. Click **Rules**.

The notebook of the selected action opens. If you selected several actions, the notebook of the first-selected action opens.

On the Rules page, you see the rules that can be applied to carry out the selected action. By default, all the rules are selected.

Figure 7 shows the Action notebook for the creation of indexes.

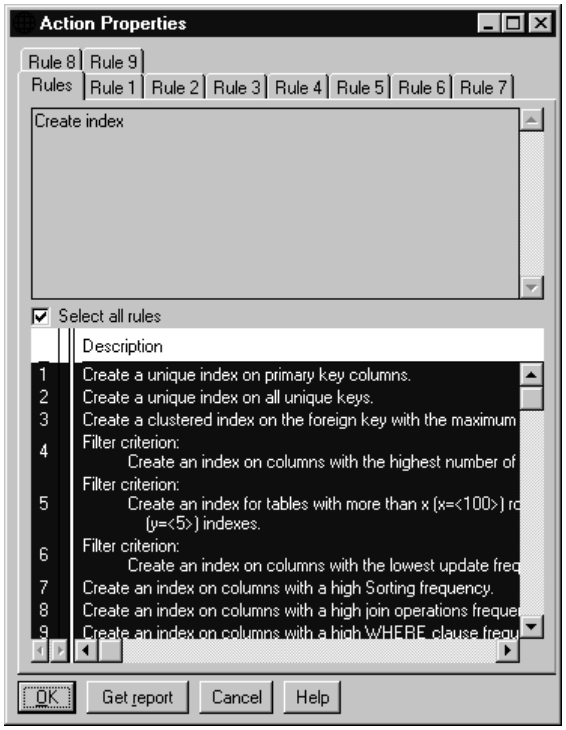


Figure 7. Settings Notebook for the Creation of Indexes

3. Deselect the rules you don't want to include in the action.
4. Open the notebook page of a rule.

   You see the detailed description of the rule.
5. Where appropriate, change the necessary parameters.
6. Click **OK**.

   Your settings for the selected action are saved, and you see the list of actions again.
7. Repeat these steps, if you selected several actions.

Note: If you selected several objects, your settings apply to all of the objects.

Next, you can request a report.

Requesting Design Reports

After you select a specific design action and tailor its properties, you can request a design report. For a design proposal, you can request a report for only one action at a time. For design information and validation, you can request a report for several actions.

To get a report, click **Get report** for the selected action or actions.

Evaluating Design Support Reports

Depending on the type of design support, a report contains information messages, design proposals, or warning messages. The report messages are headed by an icon. With design proposals, there is an additional column with the title **Accepted**. It indicates whether a proposal has been accepted or rejected.

A report with information on potential design problems contains information messages. A design validation report contains warning messages. A report with design proposals contains design proposals as well as information messages. Here, the information messages inform you about design information that is missing, but necessary for the creation of an appropriate proposal.

Figure 8 on page 91 shows a report with proposals and information messages for the creation of indexes.

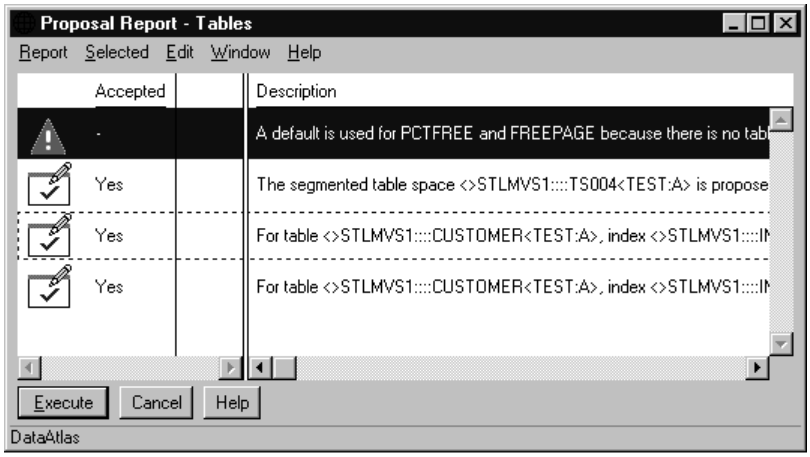


Figure 8. Proposal Report

For all types of messages, you can ask for the action item that caused the message:

1. Select a report item; then click **Selected** on the menu bar. (Or just click the item with the right mouse button).
2. Click **Rule** on either menu.

   You see the detailed description of the rule on which the action item is based:

Also, for each message, you can look at the affected objects:

1. Click a report item with the right mouse button.

   A pop-up menu appears.
2. Click **Details**.

   A window lists the report item again, and presents icons for the affected objects:
3. Double-click the icon to open the object's notebook and look at its specifications.

With a validation report item or an information report item, you only see an icon for the object you are currently designing, because it is the only one affected. With a design proposal item, you also get icons for the proposed objects.

See the next section for a detailed description of how to accept proposals.

Sometimes you see report items where you have to make a choice from several proposals (see the section “Selecting from a Choice of Proposals” on page 93 for a detailed description).

For a detailed description of how you can execute accepted design proposals, see the section “Executing Design Proposals” on page 93.

Accepting Design Proposals

You have several possibilities for accepting design proposals.

To accept design proposals from the Report Item window, do the following:

1. From a design proposal report, select a report item with the right mouse button.

   A pop-up menu appears.
2. Click **Details**.

   A window lists the report item again, and presents icons for the current and for the proposed objects.
3. Double-click the icon of the proposed object or the object with proposed changes that you want to accept.

   The object’s notebook opens. The notebook already contains the proposed values.
4. Leave the notebook by clicking **OK**.
5. Close the Report Item window.

   In the report, the corresponding report item is marked **Accepted - Yes**, and the icon gets a check mark.

To accept design proposals from the design proposals report, do the following:

1. Select a proposed item.
2. From the **Selected** pull-down menu:
   - Click **Accept** if you agree with the proposal.

     The accepted proposals are marked **Accepted - Yes**, and the icon gets a check mark.
   - Click **Reject** if you do not agree with the proposal.

     The rejected proposals are marked **Accepted - No**.

Figure 9 on page 93 shows a report with accepted and rejected design proposals.

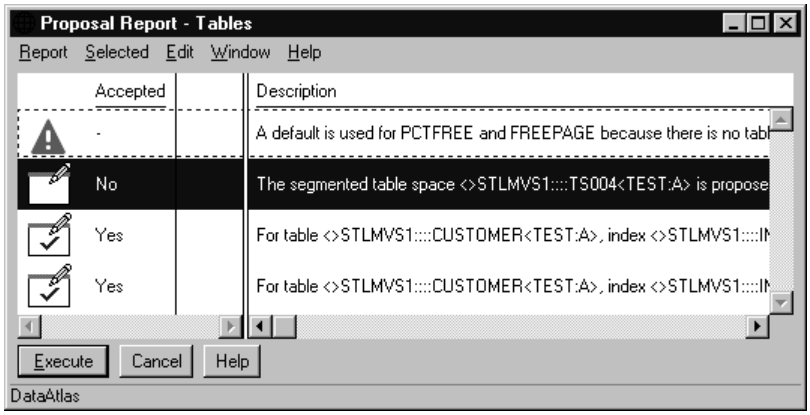


Figure 9. Accepted and Rejected Design Proposals

Selecting from a Choice of Proposals

Sometimes you have to choose between various proposals:

1. Click the multiple-proposal report item with the right mouse button.

   A pop-up menu appears.
2. Click **Details**.

   A window lists the report item again, and presents the icons of the objects to choose from.
3. Select the appropriate object.

   The corresponding notebook opens.
4. Click **OK** to accept and save the proposed values.

   The notebook closes, and you are back on the Report Item window.
5. Click **Cancel** to leave the Report Item window.

   You see the report again. The report item is marked **Accepted - Yes**, and the icon gets a check mark.

For a detailed description of how you can execute design proposals from the design proposals report, see the next section.

Executing Design Proposals

If you want to save the proposed changes to objects or the proposed new objects, do the following:

1. Mark the items you want to save with **Accepted - Yes**.

   (See the previous sections for how to accept proposed items.)

2. Click **Execute**.

   The proposals that are marked **Accepted - Yes** are executed.

   After execution, the proposals are marked **Executed**. If new objects are created, a window will ask you whether you want the objects to be represented in a workfolder.

Figure 10 shows a report with executed design proposals.

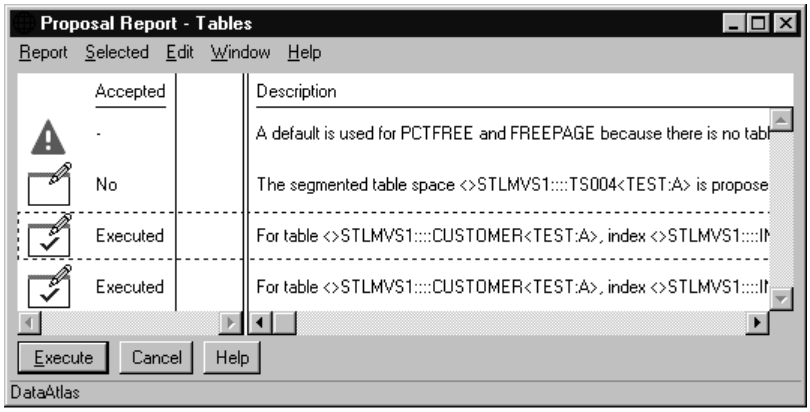


Figure 10. Executed Design Proposals

Chapter 13. Designing Tables

The physical design is carried out with notebooks. The notebook sections indicate the different design areas.

You perform the physical design by filling in the necessary fields on the notebook pages. In addition, DataAtlas Designer offers design support. For the table object, all the types of design support are available:

- Information on potential design problems
- Design proposals
- Design validations

The design of the table object consists of the following tasks:

- Specifying general information
- Viewing the shareable table definition
- Viewing the columns
- Adding and deleting columns
- Optimizing the table layout
- Assigning the table to a physical design
- Specifying routines and options
- Creating and modifying a primary key
- Creating and modifying unique keys
- Creating and modifying foreign keys
- Creating indexes
- Assigning the table to a table space and a database
- Creating table partitions
- Modifying table partitions
- Specifying the data load
- Specifying the work load
- Extracting DB2 actual values

Below, you find a detailed description of these tasks.

Available Design Support for Tables

For the table object, the following design support is available:

- Inform — Information on potential design problems:
  - Identification of tables with default values
  - Identification of tables with potential column problems
  - Identification of tables with potential primary key problems
  - Identification of tables with missing design information
  - Identification of tables without an explicit clustered index
  - Identification of large tables
  - Identification of columns with missing design information
  - Calculation of wasted space
- Propose — Design proposals:
  - Assignment of the table to a table space
  - Creation of a primary key
  - Creation of foreign keys
  - Creation of indexes
  - Creation of a partitioned index for a partitioned table
- Validate — Design validations:
  - Identification of tables with invalid primary or foreign keys
  - Identification of tables with column inconsistencies
  - Identification of incomplete tables

Refer to the section “Selecting Design Actions” on page 88, for a description of how to request design support for the listed areas of design.

Specifying General Information

To specify the general information of the table object, do the following:

1. Open the Table notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
   - Name
   - Variation

- Revision
- Component
- Description

Some fields are mandatory, others optional.

Viewing the Shareable Table Definition

To view the shareable table definition, do the following:

1. Open the Table notebook to the Definition page.
2. Click **Open** to open the notebook of the table definition.

The column information related to the table definition is listed in the **Columns** table.

Note: When you use a shareable table definition, you must be careful: When you change the column definitions, the changes are propagated to all of the table objects using this shareable table definition.

Viewing the Columns

To view all of the columns of a table, do the following:

1. Open the Table notebook to the Definition page.
   You see the columns defined for the table listed in the **Columns** table.
2. Click **Open** to open a column's notebook and look at its characteristics.

Adding and Deleting Columns

To add a new column to the table, do the following:

1. Open the Table notebook to the Definition page.
2. Click **Create** from the **Columns** table.
   An empty column notebook opens.
3. If you want to use shareable data elements, click **Search** on the Definition page of the Column notebook.
4. To disassociate a shareable data element from a column, click **Remove**. The data element's name is cleared from the entry field. Values in the data definition fields are also cleared.

The specified information is listed in the **Columns** table.

For the design of the column, refer to the section “Chapter 14. Designing Columns” on page 107 for a detailed description.

To delete a column from the table, do the following:

1. Open the Table notebook to the Definition page.
2. Select a column from the **Columns** table.
3. Click **Delete**.

The column is deleted from the table object, and its information is removed from the **Columns** table.

Note: When you change the column information of a table, it causes a change in the shareable table definition, too.

Optimizing the Table Layout

To optimize the table layout, do the following:

1. Open the Table notebook to the Definition page.

   The table’s columns and their characteristics are listed in the **Columns** table.
2. Select the column you want to reposition.
3. Click **Move up**.

   The column is moved up in the table.
4. Click **Move down**.

   The column is moved down in the table.

Assigning the Table to a Physical Design

Only when an object is assigned to a physical design, can you fully exploit the design support functions available for the entire physical design. For example, from the physical design, you can request design proposals for several objects at a time.

To assign the table object to a physical design, do the following:

1. Open the Table notebook to the Designs page.
2. Click **Search** to search for an appropriate physical design.
3. Select a physical design from the resulting list.

Specifying Routines and Options

To specify routines and set the options for the table object, do the following:

1. Open the Table notebook to the Options page.
2. Specify the following optional information in the corresponding fields:
   - Procedures
     - EDITPROC
     - VALIDPROC
   - OBID
   - Audit
     - None
     - Changes
     - All
   - Data capture
     - None
     - Changes
   - Label
   - Comment

     (Use this field for your own notes and remarks on the table object.)

Creating and Modifying a Primary Key

The primary key clearly identifies the rows of a table. It can consist of one or more columns. Only unique values are allowed in primary key columns.

For the creation of a primary key, you have the choice between two different design modes:

- Directly performing this design step in the notebook
- Getting a design proposal from DataAtlas Designer

If you want to directly create the table's primary key, do the following:

1. Open the Table notebook to the Primary Key page.
2. From the **Available** list box, select the columns of which the primary key should be composed.
3. Click **Add**.

The selected columns are listed in the **Selected** list box. The primary key is composed of these columns.

Or, to request a design proposal for the creation of a primary key, click **Propose** on the **Designer** pull-down, and then **Create primary key**.

Note: The primary key is a unique key. If you assign unique keys, you must also assign unique indexes. Otherwise you cannot use the implemented table.

When you want to modify the primary key, do the following:

1. From the **Selected** list box, select the columns you want to remove from the primary key.
2. Click **Remove**.

The selected columns are not part of the primary key any more.

Note: You cannot modify a primary key that is referenced by a foreign key.

Creating and Modifying Unique Keys

Like primary keys, unique keys are a way of allowing only unique values in a column.

To create unique keys, do the following:

1. Open the Table notebook to the Unique Keys page.
2. From the **Available** list box, select the columns of which the unique key should be composed.
3. Click **Add**.

   The selected columns are listed in the **Selected** list box.
4. Click **Remove** to remove selected columns from the **Selected** list box.
5. Click **Create**.

   The columns of the **Selected** list box are listed as unique key in the **Unique keys** list box, separated by an exclamation mark (!). Each line represents one unique key.

If you want to modify a unique key, do the following:

1. Select a unique key from the **Unique keys** list box.

   The corresponding unique key columns are listed in the **Selected** list box.
2. Modify the set of unique key columns by adding or removing columns.
3. Click **Modify**.

   The changed set of columns is listed as unique key in the **Unique keys** list box.

4. Click **Delete** if you want to delete a selected unique key from the **Unique keys** list box.

Note: If you assign unique keys, you must also assign unique indexes. Otherwise you cannot use the implemented table.

Creating and Modifying Foreign Keys

The foreign key is used to specify a referential constraint on the table. You can create more than one foreign key on the table.

Parent table primary key columns and foreign key columns of the current, dependent table must match in the following aspects:

- Number of columns
- Data types
- Field procedures

For the creation of foreign keys, you have the choice between two different design modes:

- Directly performing this design step in the notebook
- Getting a design proposal from DataAtlas Designer

If you want to create a foreign key directly, do the following:

1. Open the Table notebook to the Foreign Keys page.
2. Click **Search** to search for the parent table whose primary key corresponds to the foreign key that you want to create.

   The primary key columns of the referenced parent table are listed as **Parent table primary key columns**. The columns of the current table whose data type values match the ones of the referenced key columns are listed as **Matching columns**.
3. Select the matching columns that should compose the foreign key.
4. Specify a name in the **Constraint name** entry field.
5. Specify the type of delete rule for the newly specified key.
6. Click **Create**.

   The newly specified foreign key is listed in the **Constraint list** list box.
7. Repeat these steps if you want to create further foreign keys.

If you want to modify a foreign key, do the following:

1. Select a foreign key from the **Constraint list** list box.

All of the columns of the table are listed in the **Matching columns** list box. The current columns of the selected foreign key are marked.

2. Modify the set of foreign key columns by marking different columns.
3. Change the constraint name if appropriate.
4. Click **Modify**.

   The constraint with its changed set of columns is listed in the **Constraint list** list box.

If you want to request a design proposal for the creation of foreign keys, click **Propose** on the **Designer** pull-down, and then **Create foreign keys**.

Creating Indexes

Create indexes to optimize the access path.

For the creation of indexes, you have the choice between two different design modes:

- Directly performing this design step in the notebook
- Getting a design proposal from DataAtlas Designer

If you want to create indexes directly, do the following:

1. Open the Table notebook to the Indexes page.

   In the **Indexes** list box, you see all of the table's current indexes.
2. Click **Create**.

   An empty index notebook displays into which you can enter the required data. For a detailed description, refer to the section "Chapter 15. Designing Indexes" on page 111. After closing the index notebook, the new index is also listed in the **Indexes** list box.

To change an index, do the following:

1. Select an index from the **Indexes** list box.

   The columns belonging to the selected index are displayed in the **Columns of selected index** list box.
2. Click **Open** to open the corresponding notebook and change the necessary data.

If you want to request a design proposal for the creation of indexes, click **Propose** on the **Designer** pull-down, and then click **Create index**.

Assigning the Table to a Table Space and a Database

A table must be stored in an appropriate table space. Assign the table explicitly to a table space to avoid that the DB2 default assignments are used.

For the assignment of the table to a table space, you have the choice between two different design modes:

- Directly performing this design step in the notebook
- Getting a design proposal from DataAtlas Designer

If you want to directly assign the table to a table space, do the following:

1. Open the Table notebook to the Table Space page.
2. Click **Search** to select an appropriate table space from the list of existing table spaces.

The selected table space appears in the corresponding entry field. If the table space is assigned to a database, this database is automatically assigned to the table, too, and the name of the database is listed in the corresponding entry field.

If the selected table space is not assigned to a database, no database is directly assigned to the table. DB2 then assigns the table to the default database DSNDB04.

If you do not want to explicitly assign the table to a table space, you can also first select the database. If the database contains exactly one table space, this table space is listed in the table space entry field. If the database contains more than one table space, the table space entry field remains empty. If you do not explicitly select one of the assigned table spaces in the course of your physical design, the DB2 default table space is used then. The DB2 default table space is also used, if the database has no table space assigned.

If you want to request a design proposal for the table to table space assignment, click **Propose** on the **Designer** pull-down, and then **Assign table to table space**.

Creating Table Partitions

To optimize the performance, large tables are best assigned to partitioned table spaces. To support the table's assignment to a partitioned table space, DataAtlas Designer lets you create table partitions. Only, if you created table partitions, you can specify a partitioned table space or a partitioned index.

DataAtlas Designer's design support function also only proposes a partitioned table space or a partitioned index for the table if you created table partitions before.

Note: Table, table space, and index partitions should always match in number.

You create table partitions like this:

1. Open the Table notebook to the Partitions page.
2. Specify the number of partitions you want to create.
3. Click **Accept**.

   The numbered partitions are listed in the **Partitions** table.
4. From the **Available** list box, select the columns for the partition.
5. Click **Add**.

   The selected columns are listed in the **Selected** list box. In addition, they are inserted into the **Partitions** table.
6. Select a partition from the **Partitions** table.
7. Click **Open** to specify range values for the columns of the partition.

   A Table Partitions window displays. The number of the partition is listed. The names of the columns for which you create ranges are inserted automatically.
8. For each column, specify the target UPPER value delimiting its range.

   The concatenation of specified column ranges determines the partition.
9. Close the Table Partitions window by clicking **OK**.

   The partition and its specified column ranges are listed in the **Partitions** table.

You can modify a partition at any time (see the corresponding section below).

Note: Be aware that DB2 uses only the first 40 bytes of the concatenated range values. The bytes exceeding this range are ignored.

Modifying Table Partitions

Changes you make to the table partitions do not affect DDL generation. If you want to keep the table partitions as a source for the table space and index partitions, you must either change the partitioned table space and partitioned index manually or request a design proposal for the creation of a new table space and index.

To modify a table partition, do the following:

1. Open the Table notebook to the Partitions page.
2. From the **Partitions** table, select the partition you want to modify.
3. Click **Open**.

   The Table Partitions window opens. You see the number of the partition and the current target UPPER values delimiting the ranges of the columns.
4. Change the range values where necessary.
5. Close the Table Partitions window.

   The changes are listed in the **Partitions** table.

Specifying the Data Load

Together with the work load values, the data load values determine the adequate configuration of a table's storage and access structures. Many of the design functions DataAtlas Designer offers make use of these values. The most important value here is the *initial number of rows* value.

To specify the data load of the table, do the following:

1. Open the Table notebook to the Data Load page.

   You see a read-only table with the data load for the listed partitions.
2. Select a partition.
3. Click **Change**.

   The Table Partitions Data Load window opens.
4. Enter the necessary data load information.
5. Click **OK** to close the window.

   Your specifications are listed in the data load table.

The data load values are required, for example, for the calculation of index proposals.

For a detailed description of the fields, see DataAtlas online help.

Specifying the Work Load

Together with the data load values, the work load values determine the adequate configuration of a table's storage and access structures.

The work load information for a table consists in specifications of frequency and concurrency values for a number of operations.

To specify the work load, do the following:

1. Open the Table notebook to the Work Load page.

   You see a read-only table with the work load for the listed partitions.
2. Select a partition.
3. Click **Change**.

   The Table Partition Work Load window opens.
4. Specify the necessary frequency and concurrency data.
5. Click **OK** to close the window.

   Your specifications are listed in the work load table.

The work load values are required, for example, for the calculation of index proposals.

For a detailed description of the fields, see DataAtlas online help.

Extracting DB2 Actual Values

Here you extract the most important DB2 statistic values of the table from the DB2 catalog. Together with the statistic values for the table, the statistic values of all columns and assigned indexes are retrieved.

To extract the statistic values of the table object, do the following:

1. Open the Table notebook to the DB2 Actuals page.
2. Click **Search** to look for the database that contains the DB2 catalog from which you want to extract the statistic values.
3. Click **Extract statistics** to extract the DB2 statistic values of the table from the DB2 catalog.

   The values are listed in the corresponding fields.

For a description of the fields, see DataAtlas online help.

Note: No data is available for extraction as long as the object has not been implemented.

Chapter 14. Designing Columns

The physical design of the column object is carried out with the column notebook. It is accessed from the Definition page of the Table notebook.

Design advice for the column object is given in terms of information on potential design problems.

The design of the column object consists of the following tasks:

- Defining a column
- Setting options
- Specifying the data load
- Specifying the work load
- Viewing DB2 actual values

Below, you find a detailed description of these tasks.

Available Design Support for Columns

For the column object, the following design support is available:

- Inform — Information on potential design problems:
  - Columns with data type problems
  - Columns with missing design information

No Propose or Validate functions are available for the column object.

Refer to the section “Selecting Design Actions” on page 88 for a description of how to request design support for the listed areas of design:

Note: Since columns are part of a table, their design actions are also part of the ones for the corresponding table.

Defining a Column

To specify the parameters by which the column is defined:

1. Do one of the following on the Definition page of the Table notebook:
   - Click **Create** if you want to create a new column.

     An empty column notebook displays.
   - Select a column and click **Open** if you want to work with an existing column.

     The corresponding notebook opens.
2. Open the Definition page of the Column notebook.
3. Specify the listed parameters:
   - Name

     The **Name** is mandatory.

     You can change the name of a column at any time.
   - Owning table definition

     This field is read-only. It shows the table definition that the column belongs to.
   - Shareable data element

     If you want to use predefined shareable data elements, click **Search** to search for an appropriate one.

     Click **Open** to open the notebook of the shareable data element.

     Click **Remove**. The data element's name is cleared from the entry field. Values in the data definition fields are also cleared.
   - Data type
     - Precision
     - Scale
     - Length
     - Byte count
   - Subtype:
     - Bit
     - Mixed
     - SBCS
     - None

A shareable data element has its own data type. Therefore, if the column is based on a shareable data element, the **Data type** fields are filled and read-only. They can only be changed in the notebook of the shareable data element.

For a description of the fields, see DataAtlas online help.

Setting Options

To specify options for the column object, do the following:

1. Open the Column notebook to the Options page.
2. Specify the following values:
   - Field Procedure:
     - Name
     - Parameters
   - Allow nulls:
     - Yes
     - No
     - Default
     - Unique

The **Comment** field gives you room for your own notes and comments on the object you are currently designing.

After the table has been implemented, the field procedure for the table cannot be changed, added, or deleted.

The field procedure is optional. If you omit it, the column has no field procedure.

For a detailed description of the fields, see DataAtlas online help.

Specifying the Data Load

Together with the work load values, the data load values of the column determine the adequate configuration of the storage and access structures of the corresponding table.

1. Open the Column notebook to the Data Load page.
2. Enter the necessary data load information.

The data load values are required, for example, for the calculation of index proposals.

For a description of the fields, see DataAtlas online help.

Specifying the Work Load

Together with the data load values, the work load values of the column determine the adequate configuration of the storage and access structures of the corresponding table.

1. Open the Column notebook to the Work Load page.
2. Enter the necessary work load information.

The work load values are required, for example, for the calculation of index proposals.

For a description of the fields, see DataAtlas online help.

Viewing DB2 Actual Values

Statistic values of a column can be extracted through the **Extract statistics** function on the DB2 Actuals page of the Table notebook.

To view these values, open the Column notebook to the DB2 Actuals page. You see the most important statistic values of the column from the DB2 catalog.

For a description of the fields, see DataAtlas online help.

Chapter 15. Designing Indexes

The physical design is carried out with notebooks. The notebook sections indicate the different design areas.

You perform the physical design by filling in the necessary fields on the notebook pages. In addition, DataAtlas Designer offers design support. For the index object, all the types of design support are available:

- Information on potential design problems
- Design proposals
- Design validations

The design of the index object consists of the following tasks:

- Specifying general information
- Assigning the index to a physical design
- Defining an index
- Specifying index columns
- Specifying the sorting order
- Specifying the storage information
- Viewing and modifying index partitions
- Specifying type and storage information of an index partition
- Viewing DB2 actual values

Below, you find a detailed description of these tasks.

Available Design Support for Indexes

For the index object, the following design support is available:

- Inform — Information on potential design problems:
  - Identification of indexes that can result in increased maintenance cost
  - Identification of indexes that could be dropped
- Propose — Design proposals:
  - Calculation of space requirements
  - Setting of options
- Validate — Design validations:
  - Identification of indexes with column inconsistencies

Refer to the section “Selecting Design Actions” on page 88 for a description of how to request design support for the listed areas of design.

Specifying General Information

To specify the general information of the index object, do the following:

1. Open the Index notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
     - Database
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Assigning the Index to a Physical Design

Only, when an object is assigned to a physical design, you can fully exploit the design support functions available for the entire physical design. For example, you can request design proposals for several objects at a time from the physical design.

To assign the index object to a physical design, do the following:

1. Open the Index notebook to the Designs page.
2. Click **Search** to search for an appropriate physical design.
3. Select a physical design from the resulting list.

Defining an Index

To specify the parameters by which the index is defined, do the following:

1. Open the Index notebook to the Definition page.
2. Specify the listed parameters:
   - Owning table

Click **Open** to open the notebook of the owning table. Click **Search** to search for the table on which you want to define the index.

- Password
- Subpages
- Buffer pool
- Defer
- Close
- Organization
  - Unique
  - Clustered
    - Partitioned
      - Number of partitions

Note: You cannot specify a partitioned index if the owning table is not partitioned.

For a detailed description of the fields, see DataAtlas online help.

Specifying Index Columns

To specify the columns on which you want to define an index, do the following:

1. Open the Index notebook to the Columns page.
2. From the **Available Columns** list box, select the columns on which you want to define the index.
3. Click **Add** to add the selected columns to the index.

   The selected columns are added to the **Selected Columns** list box.
4. Click **Remove** if you want to remove selected columns from the index.

   The selected columns are dropped from the **Selected Columns** list box.

Specifying the Sorting Order

To specify the sorting order of index columns, do the following:

1. Open the Index notebook to the Columns page.
2. From the columns in the **Selected Columns** list box, select a column.

   The type of sorting order of this column is listed. It can be either of the following orders:

   - Ascending

- Descending

3. You can change the sorting order, if needed.

Specifying Storage Information

To specify the storage information of the index, do the following:

1. Open the Index notebook to the Storage page.
2. Specify the listed storage information.

For a detailed description of the fields, see DataAtlas online help.

Viewing Index Partitions

You can only view the existing index partitions on this page. To create an index partition, switch to the notebook of the owning table and request a proposal for the creation of indexes.

Note: Only, when the table is partitioned, you can get a proposal for a partitioned index. To view the existing index partitions, do the following:

1. Open the Index notebook to the Partitions page.
2. In the **Partition definition** list box, you see how many index partitions exist, and for which columns they are defined.
3. In the **Partitions** list box, you see the type and storage information of each partition.

   Click **Open** to open the Index Partition window where you can specify the type and the storage information of each partition (see the corresponding section below).

Specifying Type and Storage Information of an Index Partition

To specify the type and the storage information of index partitions, do the following:

1. From the **Partitions** table, select the partition for which you want to specify the type and the storage information.
2. Click **Open**.

   The Index Partition window displays.
3. Fill in the listed type and storage information.
4. Close the window.

You see the values you specified in the **Partitions** table. The **Type** column shows whether you selected to manage the data sets for the index yourself (VCAT option), or whether you want DB2 to do this for you (Storage Group option).

For a detailed description of the fields, see DataAtlas online help.

Viewing DB2 Actual Values

To view the index statistic values from the DB2 catalog, open the Index notebook to the DB2 Actuals page. The current values of the index or index partitions are displayed.

The index values are extracted from the DB2 catalog through extraction of the values of the owning table.

No data is available for extraction as long as the owning table object has not been implemented.

Chapter 16. Designing Table Spaces

The physical design is carried out with notebooks. The notebook sections indicate the different design areas.

You perform the physical design by filling in the necessary fields on the notebook pages. In addition, DataAtlas Designer offers design support. For the table space object, all the types of design support are available:

- Information on potential design problems
- Design proposals
- Design validations

The design of the table space object consists of the following tasks:

- Specifying general information
- Assigning the table space to a physical design
- Assigning the table space to a database
- Setting options
- Creating table space partitions
- Specifying type and storage information of a table space partition
- Specifying storage information
- Assigning tables
- Selecting a buffer pool
- Specifying design information
- Extracting DB2 actual values

Below, you find a detailed description of these tasks.

Available Design Support for Table Spaces

For the table space object, the following design support is available:

- Inform — Information on potential design problems:
  - Identification of table spaces with default values
  - Identification of table spaces with design inconsistencies
- Propose — Design proposals:
  - Calculation of space requirements
  - Calculation of SEGSIZE

  - Setting options
- Validate — Design validations:
  - Identification of invalid table spaces assigned to DSNDB07
  - Identification of invalid table spaces
  - Identification of incomplete table spaces

Refer to the section “Selecting Design Actions” on page 88 for a description of how to request design support for the listed areas of design.

Specifying General Information

To specify the general information of the table space object, do the following:

1. Open the Table Space notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
     - Database
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Assigning the Table Space to a Physical Design

Only, when an object is assigned to a physical design, you can fully exploit the design support functions available for the entire physical design. For example, you can request design proposals for several objects at a time from the physical design.

To assign the table space object to a physical design, do the following:

1. Open the Table Space notebook to the Designs page.
2. Click **Search** to search for an appropriate physical design.
3. Select a physical design from the resulting list.

Assigning the Table Space to a Database

To assign the table space to a database, do the following:

1. Open the Table Space notebook to the General page.
2. Click **Search** with the database entry field to search for an existing database to which you can assign the table space.

The database is listed in the corresponding entry field.

Note: For a detailed description of the search procedure, refer to the corresponding section in the *DataAtlas Dictionary User's Guide* or to DataAtlas online help.

Setting Options

To set the options for a table space object, do the following:

1. Open the Table Space notebook to the Options page.
2. Specify the following values:
   - Type
     - Simple
     - Segmented
       - Segment size
     - Partitioned
       - Number of partitions
   - Locking size
   - Password
   - Close Option

For a detailed description of the fields, see DataAtlas online help.

Creating Table Space Partitions

To create table space partitions, do the following:

1. Open the Table Space notebook to the Partitions page.
2. Click **Create**.

   The Table Space Partition window opens.
3. Specify the running number of the partition that you want to edit.
4. Specify the necessary type and storage information.

5. Click **OK**.

   You are back on the Partitions page and the values you specified are listed in the Partitions table.

For a detailed description of the fields, see DataAtlas online help.

Note: With a partitioned table space, you must specify a partitioned index for the table that is assigned to the table space.

If you do not specify a partition's information on using block, free block, or compress, the table space's values as specified on the Storage page also hold for the partition.

Specifying Type and Storage Information of a Table Space Partition

To specify the type and the storage information of a table space partition, do the following:

1. Open the Table Space notebook to the Partitions page.
2. From the partitions table, select the partition whose information you want to specify.
3. Click **Open**.

   The Table Space Partition window opens.
4. Specify the necessary type and storage information.
5. Click **OK**.

   You are back on the Partitions page and the values you specified are listed in the Partitions table.

For a detailed description of the fields, see DataAtlas online help.

Note: If you do not specify a partition's information on using block, free block, or compress, the table space's values as specified on the Storage page also hold for the partition.

Specifying Storage Information

To specify the storage information of the table space, do the following:

1. Open the Table Space notebook to the Storage page.
2. Specify the necessary type and storage information.

For a detailed description of the fields, see DataAtlas online help.

Assigning Tables

To assign tables to the table space, do the following:

1. Open the Table Space notebook to the Tables page.
2. Click **Search** to search for existing tables to assign to the table space.
3. Click **Remove** to remove selected tables from the present table space.

The assigned tables are listed in the list box.

Selecting a Buffer Pool

To select a buffer pool for the table space, do the following:

1. Open the Table Space notebook to the Buffer Pool page.
2. From the list box, select the appropriate buffer pool for the table space.
   The selected buffer pool is listed in the **Buffer pool** entry field.

Specifying Design Information

To specify the design information for the table space, do the following:

1. Open the Table Space notebook to the Design Info page.
2. Specify the following values:
   - Keep in main storage
   - Usage intent

For a detailed description of the fields, see DataAtlas online help.

Extracting DB2 Actual Values

Here, you extract the most important DB2 statistic values of the table space from the DB2 catalog. Together with the statistic values for the table space, the statistic values of the assigned storage group are retrieved.

To extract the statistic values of the table space object, do the following:

1. Open the Table Space notebook to the DB2 Actuals page.
2. Click **Search** to look for the database that contains the DB2 catalog from which you want to extract the statistic values.
3. Click **Extract statistics** to extract the DB2 statistic values of the table space from the DB2 catalog.

Note: No data is available for extraction as long as the table space has not been implemented.

Chapter 17. Designing Databases

The physical design is carried out with notebooks. The notebook sections indicate the different design areas.

You perform the physical design by filling in the necessary fields on the notebook pages. In addition, DataAtlas Designer offers design support. For an existing database object, you can get design support in terms of information on potential design problems.

The design of the database object consists of the following tasks:

- Specifying general information
- Setting options
- Specifying design information
- Viewing assigned tables spaces
- Assigning the database to a storage group
- Selecting a buffer pool
- Assigning the database to a physical design

Below, you find a detailed description of these tasks.

Available Design Support for Databases

For a database object, the following design support is available:

- Inform — Information on potential design problems:
  - Identification of databases with default values
  - Identification of databases with performance problems

No Propose or Validate functions are available for a database object.

To get design support, click **Inform** on the **Designer** pull-down of the Database notebook.

Specifying General Information

To specify the general information of the database object, do the following:

1. Open the Database notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Setting Options

To set the options for a database object, do the following:

1. Open the Database notebook to the Options page.
2. Specify the listed options.

For a detailed description of the fields, see DataAtlas online help.

Specifying Design Information

To specify the design information of a database object, do the following:

1. Open the Database notebook to the Design Info page.
2. Specify the listed design information.

For a detailed description of the fields, see DataAtlas online help.

Viewing Assigned Table Spaces

To view the table space objects assigned to a database object, do the following:

1. Open the Database notebook to the Table Spaces page.

   In the **Assigned table spaces** list box, you see the table spaces currently assigned to the database.

Click **Open** to open the notebook of a selected table space.

You cannot assign table spaces to the database on this page. To assign a table space to the database, you need to open the notebook of the table space.

Assigning the Database to a Storage Group

To assign the database to a storage group, do the following:

1. Open the Database notebook to the Storage Group page.
2. Click **Search** to search for an existing storage group to which you can assign the database.

Selecting a Buffer Pool

To select a buffer pool for the database object, do the following:

1. Open the Database notebook to the Buffer Pool page.
2. From the list box, select the appropriate buffer pool for the database.
   The selected buffer pool is listed in the **Buffer pool** entry field.

Assigning the Database to a Physical Design

To assign the database to a physical design, do the following:

1. Open the Database notebook to the Designs page.
2. Click **Search** to search for an existing physical design.
3. Select a physical design from the resulting list.

Chapter 18. Designing Storage Groups

The physical design is carried out with notebooks. The notebook sections indicate the different design areas.

You perform the physical design by filling in the necessary fields on the notebook pages. In addition, DataAtlas Designer offers design support. For the storage group object, all the types of design advice are available:

- Information on potential design problems
- Design proposals
- Design validations

The design of the storage group object consists of the following tasks:

- Specifying general information
- Assigning the storage group to a physical design
- Specifying storage information
- Assigning databases
- Viewing assigned table spaces
- Viewing assigned indexes
- Specifying the usage intent
- Specifying the required space
- Converting the used space value
- Viewing DB2 actual values

Below, you find a detailed description of these tasks.

Available Design Support for Storage Groups

You can get design support from anywhere in the notebook.

For the storage group object, the following design support is available:

- Inform — Information on potential design problems:
  - Identification of storage groups with SMS usage
- Validate — Design validations:
  - Identification of storage groups with invalid definitions
  - Identification of incomplete storage groups
- Propose — Design proposals:

– Calculation of required space

To get design support in any of these design areas, click the appropriate design function on the **Designer** pull-down of the Storage Group notebook.

Specifying General Information

To specify the general information of the storage group object, do the following:

1. Open the Storage Group notebook, page General.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     – System
     – Creator
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Assigning the Storage Group to a Physical Design

To assign the storage group to a physical design, do the following:

1. Open the Storage Group notebook to the Designs page.
2. Click **Search** to search for an existing physical design.
3. Select a physical design from the resulting list.

Specifying Storage Information

For the specification of the storage information of a storage group, do the following:

1. Open the Storage Group notebook, page Storage.
2. Enter the volume catalog (VCAT) name in the provided entry field.
3. Specify whether the volume catalog is password protected, and enter the password, if necessary.
4. Specify the kind of device type.

5. Select one of the following types of storage management:
   - Storage management by the system (SMS)
   - Storage management by the user

   If you decide to manage the storage yourself, specify the volumes:
   a. Click **Add volume** to add the volume specified in the entry field to the list of available volumes.
   b. Click **Delete volume** to delete the volume specified in the entry field from the list of available volumes.
   c. Click **Add** to add selected volumes from the list of available volumes to the list of volumes selected for allocation.
   d. Click **Remove** to remove selected volumes from the list of volumes selected for allocation.

Assigning Databases

To assign databases to a storage group object, do the following:

1. Open the Storage Group notebook to the Databases page.
2. Click **Search** to search for existing databases to assign to the storage group.
3. Click **Remove** to remove selected databases from the present storage group.

The currently assigned databases are listed in the list box.

Viewing Assigned Table Spaces

To view the table space objects assigned to a storage group object, open the Storage Group notebook to the Table Spaces page. You see a list box with the table spaces currently assigned to the storage group.

You cannot assign any table spaces to the storage group on this page. To assign a table space to the storage group, you need to open the notebook of the table space.

Viewing Assigned Indexes

To view the index objects assigned to a storage group object, open the Storage Group notebook to the Indexes page. You see a list box with the indexes currently assigned to the storage group.

You cannot assign any indexes to the storage group on this page. To assign an index to the storage group, you need to open the notebook of the index.

Specifying the Usage Intent

To specify the usage intent of a storage group, do the following:

1. Open the Storage Group notebook to the Design Info page.
2. Specify one of the following:
   - Table spaces (general)
   - Indexes (general)
   - Table spaces
   - Indexes
   - None

   Click **Search** to search for the appropriate table space or index.

Specifying the Required Space

To specify the space required by a storage group, do the following:

1. Open the Storage Group notebook to the Design Info page.
2. Fill in the entry field for required space.
3. Click **Convert** if you want to see the required space value converted from kilobytes to the number of cylinders or tracks.

Note: On request, DataAtlas Designer calculates the required space value and offers a proposal.

For an optimum proposal, you need to have assigned all table spaces and indexes. Also, the following table data load specifications are needed:

- Initial number of rows
- Confidence factor

Converting the Used Space Value

The amount of space used by a storage group is given in terms of kilobytes.

To see this value converted from kilobytes to the number of cylinders or tracks, do the following:

1. Open the Storage Group notebook to the DB2 Actuals page.

2. Click **Convert**.

   The Conversion Utility window is displayed.

3. Choose the appropriate device type.

   You see the used space value converted from kilobytes to the number of cylinders or tracks.

Viewing DB2 Actual Values

To view the storage group statistic values from the DB2 catalog, open the Storage Group notebook to the DB2 Actuals page. The current values are displayed.

The storage group values are extracted through extraction of the values of one of the table spaces assigned to the storage group.

If the table spaces have not been implemented yet, no extraction data is available.

Chapter 19. Designing Views

The physical design of the database design objects is carried out with notebooks. The notebook sections indicate the different design areas.

The design of the view object consists of the following tasks:

- Specifying general information
- Defining a view
- Assigning the view to a physical design

Below, you find a detailed description of these tasks.

Specifying General Information

To specify the general information of the view object, do the following:

1. Open the View notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
     - Database
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Defining a View

To define a view object, do the following:

1. Open the View notebook to the Definition page.
2. Specify the following items:
   - SQL select statement (Subselect)
   - Check option

- Comment
- Label

3. Optionally, you can list the participating (View) Columns.

For a detailed description of the fields, see DataAtlas online help.

Assigning the View to a Physical Design

To assign the view to a physical design, do the following:

1. Open the View notebook to the Designs page.
2. Click **Search** to search for an existing physical design.
3. Select a physical design from the resulting list.

Chapter 20. Designing an Alias/Synonym

The physical design of the database design objects is carried out with notebooks. The notebook sections indicate the design areas of an object.

The design of the alias/synonym object consists of the following tasks:

- Specifying general information
- Defining an alias/synonym
- Assigning the alias/synonym to a physical design

Below, you find a detailed description of these tasks.

Specifying General Information

To specify the general information of the alias/synonym object, do the following:

1. Open the Alias/Synonym notebook to the General page.
2. Specify the listed general information items:
   - Relational Database Qualifiers:
     - System
     - Creator
   - Name
   - Variation
   - Revision
   - Component
   - Description

Some fields are mandatory, others optional.

Defining an Alias/Synonym

To define an alias/synonym object, do the following:

1. Open the Alias/Synonym notebook to the Definition page.
2. Specify one of the following:
   - Synonym
   - Alias

- Comment
- Label

3. Select the type of your database management system from the list box.

Your choice of alias/synonym and type of database management system determines your further specification:

- Synonym (or Alias) for either of the following objects:
  - Table
  - View

  Click **Search** to get a list of table or view objects that can be entered into the corresponding field. If you select an object from the list, it will appear in the field.

  Click **Clear** to clear the corresponding entry field so that you can search for another object.

For a detailed description of the fields, see DataAtlas online help.

Assigning the Alias/Synonym to a Physical Design

To assign the alias/synonym to a physical design, do the following:

1. Open the Alias/Synonym notebook to the Designs page.
2. Click **Search** to search for an existing physical design.
3. Select a physical design from the resulting list.

Appendix A. Sample Files Shipped with DataAtlas

DataAtlas is packaged with sample files that can help you get familiar with its interface and functions. Sample files are stored in a workstation subdirectory that is created when you install DataAtlas. The directory path is `<DataAtlasInstallPath>\lang\en_us\samples`, with subdirectories `celdial`, `ims`, `cobol`, `query`, and `build`. `<DataAtlasInstallPath>` is the root directory where DataAtlas is installed.

DB2 UDB Sample Tables

Sample files are provided to create the DA_CELD database and to put data into its tables, views, and indexes. CELDIAL.CMD is used to create the DA_CELD database. See "Installing DataAtlas" on the CD-ROM's Electronic Showcase for more information about how to create the DA_CELD database.

The DA_CELD database tables are:

CUSTOMER
CUSTSHIP
LEADS
CUSTNO
PRODSHOW
PRODUCT
PRODINV
ORDER_SUMMARY
ORDER_DETAIL
ORDER_HIST_SUM
ORDER_HIST_DET
AR_HISTORY
COMPONENTS
PRODCOMP

IMS Sample Files

These files are IMS DBDs and PSBs. The file extension shows which is which:

B00INP01.DBD (GSAM)
B00OUT01.DBD (GSAM)
BE3ORDER.DBD (HIDAM)
BE3ORDRX.DBD (primary index)
BE3PARTS.DBD (HDAM)
BE3PSID1.DBD (secondary index)
HLPILMN.DBD (logical DBD)
HLPIMAN.DBD (HIDAM)
HLPINDX.DBD (primary index)
HLPISEC.DBD (HDAM)
HLPBPSB.PSB
HLPLPSB.PSB
PE3CPPUR.PSB
PE3ORDER.PSB
PE3PARTS.PSB

The PSBs beginning with PE3 complement the B00 and BE3 DBDs. The PSBs beginning with HLP complement the HLP DBDs.

You must populate physical DBDs, then logical DBDs, and then PSBs, in that order. It is more efficient to populate index DBDs first, but not required. Therefore, you should populate your IMS objects in this order:

1. Index DBDs
2. Physical DBDs
3. Logical DBDs
4. PSBs

COBOL Sample Files

The following sample COBOL COPY files are shipped with DataAtlas Dictionary. When you populate COBOL COPY files into the TeamConnection database, they are stored as included source definitions. CUSTOMER.CPY and ORDER.CPY present the data in the DA_CELD database as COBOL objects.

CUSTOMER.CPY
ORDER.CPY

PL/I Sample Files

The following sample PL/I Include files are shipped with DataAtlas Dictionary. When you populate PL/I Include files into the TeamConnection database, they are stored as included source definitions. CUSTOMER.INC and ORDER.INC present the data in the DA_CELDP database as PL/I objects.

CUSTOMER.INC

ORDER.INC

Sample Reconcile Mapping Tables

DA_CELD.MAP

This file contains sample entries for use with the DB2 UDB and COBOL Populate scenarios.

DATATLAS.MAP

This file contains sample entries for reconciling column definitions in a DB2 UDB table, data items in a IMS DBD, and data items in a COBOL COPY file.

DA_CELDP.MAP

This file contains sample entries for reconciling data items in a PL/I include file.

Sample COBOL Command File

EWSLPGM.CMD

This file contains code to invoke IBM VisualAge for COBOL for OS/2 compiler. The only changes that can be made to this file are to modify the compiler options or to change the file to be compiled.

Sample PL/I Command File

EWSLPPG.CMD

This file contains code to invoke IBM PL/I for OS/2 compiler. The only changes that can be made to this file are to modify the compiler options or to change the file to be compiled.

Appendix B. Qualifiers for Object Names

Objects in TeamConnection that can be accessed by name must have a name that is unique within the TeamConnection family and release. The names of some objects maintained by DataAtlas may not be unique based solely on their simplest name. For example, there could be two DB2/390 tables named ACCOUNT. Usually an object's name is unique within some collection of objects. DataAtlas has created collector objects that represent these collections of objects. The simple object name can be qualified by the name of the collector object to provide a unique TeamConnection name.

The following objects are collector objects in DataAtlas Dictionary:

Relational system
A relational system represents a DB2 UDB, DB2/390, or Oracle catalog. It is a grouping mechanism for all tables and associated objects populated into TeamConnection from that catalog. A relational system object must exist in TeamConnection to represent a given catalog prior to creating new relational tables.

Creator/Schema
A creator (or schema, in DB2 UDB and Oracle) represents the user ID that originally created the associated relational objects in the catalog.

Database
A database is a relational object whose origin is the database name as defined in the catalog. The database name can be a qualifier for those relational objects associated with a specific relational database.

Table definition
The shareable table definition represents the basic definition of a relational table. Keys and columns that are part of the table definition will have names that include the access name of the owning table definition.

PSB An IMS PSB contains a set of PCBs. An IMS PSB name is used as a qualifier for an IMS PCB.

Relational design
The relational design name is used as a qualifier for a physical design object. The relational design object is used by the DataAtlas Modeler to identify a set of relational table definitions.

See "Populate Considerations When Using DataAtlas Modeler or Designer" on page 30 for more information on relational design objects.

DataAtlas Dictionary ensures that the appropriate qualifiers are included in the name of an object when it is created. Table 3 lists the qualifiers for relational objects, and the position in the name where the qualifier is specified. Table 4 on page 143 lists the qualifiers for IMS objects, and the position in the name where the qualifier is specified.

Relational Qualifiers

Relational objects that have qualifiers have the form:

```
<prefix>qualifier4:qualifier3:qualifier2:qualifier1:access-name
<variation:revision>
```

Table 3 summarizes the name qualifiers used for relational objects.

Table 3. Relational Name Qualifiers

Relational Database Object	Qualifier	Position
System	none	
Creator/Schema	System	Qualifier 4
DB2/390 storage group	System	Qualifier 4
DB2/390 ICF catalog	System	Qualifier 4
DB2/390 volume	System	Qualifier 4
DB2/390 database	System	Qualifier 4
DB2 UDB database	System	Qualifier 4
DB2/390 buffer pool	System	Qualifier 4
DB2 UDB table space	System	Qualifier 4
Oracle table space	System	Qualifier 4
DB2/390 table space	System Database	Qualifier 4 Qualifier 2
DB2/390 table	System Creator	Qualifier 4 Qualifier 3
Oracle table	System Schema	Qualifier 4 Qualifier 3
DB2/390 index	System Creator	Qualifier 4 Qualifier 3
Oracle index	System Schema	Qualifier 4 Qualifier 3

Table 3. Relational Name Qualifiers (continued)

Relational Database Object	Qualifier	Position
DB2/390 view	System	Qualifier 4
	Creator	Qualifier 3
Oracle view	System	Qualifier 4
	Schema	Qualifier 3
Alias/Synonym	System	Qualifier 4
	Creator	Qualifier 3
DB2/390 index space	System	Qualifier 4
	Creator	Qualifier 3
DB2 UDB table	System	Qualifier 4
	Creator	Qualifier 3
	Database	Qualifier 2
DB2 UDB index	System	Qualifier 4
	Creator	Qualifier 3
	Database	Qualifier 2
DB2 UDB view	System	Qualifier 4
	Creator	Qualifier 3
	Database	Qualifier 2
Table definition	none	
View definition	none	
Key definition	Table definition	Qualifier 1
Relational design	none	
Physical design	Relational design	Qualifier 1

IMS Qualifiers

IMS objects that have qualifiers have the form:

```
<prefix>qualifier2:qualifier1:access-name <variation:revision>
```

The following table summarizes the name qualifiers used for IMS objects.

Table 4. IMS Name Qualifiers

IMS Database Object	Qualifier	Position
DBD	none	
PSB	none	

Table 4. IMS Name Qualifiers (continued)

IMS Database Object	Qualifier	Position
PCB	PSB name	Qualifier 2

Prefix Qualifiers

In some cases, DataAtlas uses a prefix qualifier instead of the colon-separated qualifiers.

There are three object types that use the prefix:

Workfolder

Data element alias

Data structure alias

The prefix value for the workfolder is the TC_BECOME environment variable for the creator of the workfolder.

The prefix value for a data element alias and data structure alias depends on how these objects are used.

When a data element alias and data structure alias refer to the name of the shareable data element or shareable data structure used by a relational, IMS, COBOL, or PL/I object, the prefix value is the name of the shareable data element or shareable data structure. For example, when a column (CUSTNO) uses the shareable data element (CustomerNumber), DataAtlas creates a data element alias named `<CustomerNumber>CUSTNO<:>`.

When a data element alias refers to the names of fields and a data structure alias refers to the names of segments in an IMS DBD, the prefix varies. For a segment, the prefix of the data structure alias is the DBD name. For a field, the prefix of the data element alias is the DBD name and the segment name separated by a period (for example, `<DBDPT.SEGPN>FLDPN<:>`).

Appendix C. Using the DATATLAS.EXE Build Script

An executable file called DATATLAS.EXE is delivered with the DataAtlas product. This executable can be used by TeamConnection as a binary build script to generate data definitions. It can also be invoked from within another build script that you have written to better meet your environmental or user-specific needs. For a complete discussion of how to create or modify build scripts, see *TeamConnection User's Guide*.

This appendix describes the sample build script shipped with DataAtlas Dictionary and the steps to perform a TeamConnection build. The input parameters and return codes associated with each type of object are listed, along with any special considerations that may apply.

Sample Build Script

DATATLAS.EXE creates DB2 DDL, Oracle DDL, IMS DBD and PSB source, COBOL COPY, and PL/I include files. The resultant DDL, IMS source, COBOL COPY, and PL/I include files are automatically stored in TeamConnection. Source files can be extracted from TeamConnection and uploaded to OS/390 or distributed to networked servers for installation.

DATATLAS.EXE is located in `<DataAtlasInstallPath>\client` in OS/2 and in `<DataAtlasInstallPath>\bin` in Windows NT.

Input Parameters

The current parameter processing information that applies to all objects is as follows:

The input parameters to DATATLAS.EXE may be specified in three ways:

- On the TeamConnection builder that uses DATATLAS.EXE as a binary build script,
- On the TeamConnection part that uses the builder (these are used IN PLACE OF parameters specified for the builder),
- When invoking the build command (these are used IN ADDITION TO parameters that are specified for the builder of the part, but if there are conflicts, these are used IN PLACE OF the other parameters).

In all cases, the parameters are specified in the form:

<parameter>=<value>

Where <parameter> and <value> differ depending on the type of object that is being generated (see the following sections for details).

Note that the <parameter> must always be specified in uppercase characters while the <value> is not case-sensitive. Also, blanks should not be used within the <value> or to separate the equal sign from the <parameter> or <value>.

TeamConnection Build

After your TeamConnection administrator has set up a TeamConnection build server, follow these steps:

1. Create a builder for the sample script.
2. Create an empty file part to receive the output from the build, and associate the builder with it. This output file is only empty until you invoke the first build. Subsequent builds will overlay its contents.
3. Connect the parts that you want to build to the empty file part, creating a build tree with the output file part at the top of the tree.
4. Build the output file part at the top of the build tree.

For more information about using the TeamConnection build facility to build applications, see *TeamConnection User's Guide.*

TeamConnection DataAtlas Objects

The following TeamConnection DataAtlas objects are included in a build tree and built using the sample build scripts shipped with DataAtlas Dictionary:

DB2 UDB Objects

- DSRORDTable
- DSRORDIndex
- DSRORDView
- DSRORDTablespace
- DSRORDPhysicalDesign

DB2/390 Objects

- DSRMRDStoragegroup
- DSRMRDDatabase
- DSRMRDTablespace

- DSRMRDTable
- DSRMRDIndex
- DSRMRDView
- DSRMRDAlternateName
- DSRMRDPhysicalDesign

Oracle Objects

- DSROracleTable
- DSROracleIndex
- DSROracleView
- DSROracleTablespace
- DSROraclePhysicalDesign

IMS Objects

- DSDBD
- DSPSB

COBOL and PL/I Objects

- DSIncludedSourceDef

For IMS DBD and PSB objects, COBOL, and PL/I IncludedSourceDef objects, whenever one of these objects or one of its component objects is modified using DataAtlas, TeamConnection will only build those objects in the build tree that have been modified since the last build, unless you specify otherwise when invoking the build.

For DB2 and Oracle objects, the ability to detect changes in an object or its component objects will be implemented in a future release, so for now, you'll need to build an entire build tree, building unchanged objects along with changed objects.

DataAtlas DB2 Sample Script Settings

Input Parameters

The following information, **DDLTYPE** and **DBALIAS**, correspond with the <parameter> and the other terms correspond with the <value>. See "Input Parameters" on page 145 for general discussion on parameters.

DDLTYPE

Create | Drop | DropAndCreate | Alter

Default is Create

DBALIAS

The alias name of the database as cataloged in DB2 UDB or related product, such as DDCS/2. This parameter is only necessary for generating ALTER DDL. The TeamConnection build server must be able to access the database for which you want to generate ALTER DDL, so DB2 UDB or a related product must be installed on the build server, and the databases for which you expect to generate ALTER DDL must be cataloged.

DBALIAS has no default value.

DBUSER

When DDLTYPE=Alter, specifies the USER option that is passed to the DB2 CONNECT command to connect to the DB2 system specified by DBALIAS.

If omitted, you are prompted for a value.

DBUSING

When DDLTYPE=Alter, specifies the USING option that is passed to the DB2 CONNECT command to connect to the DB2 system specified by DBALIAS.

If omitted, you are prompted for a value.

Special Considerations

DataAtlas generates ALTER TABLE ADD FOREIGN KEY DDL when generating CREATE TABLE DDL, even if the DDL for the table that is the parent of the foreign key has not been generated yet. For this reason, the DB2 sample build script reorders the ALTER DDL at the end of the DDL output file before exiting.

However, it is more difficult to correctly reorder CREATE VIEW DDL so that it is sequentially correct with respect to the creation of VIEWS referenced by other VIEWS. You might have to manually reorder the generated CREATE VIEW DDL before installing it on your target DB2/390 or DB2 UDB database.

The output part can have multiple inputs, but they must all belong to the same technology. For example, the output part may have several DB2 UDB tables, views, and indexes, but it cannot have both a DB2 UDB table and a DB2/390 table. If the output part has an invalid combination of inputs, the build will fail.

Return Codes

0 Successful build

8 DataAtlas generate exception; generate attempts to process input file objects after the object that could not be generated

12 TeamConnection exception caused DataAtlas generate to fail

View the `Build Message` for complete details of the build.

DataAtlas Oracle Sample Script Settings

Input Parameters

DDLTYPE corresponds with the <parameter> that is referred to in the general parameters discussion. See "Input Parameters" on page 145 for more information on input parameters.

DDLTYPE

Create | Drop | DropAndCreate | CreateOrReplace | DropAndCreateOrReplace

Default is Create.

Return Codes

0 Successful build
8 DataAtlas generate exception; generate attempts to process input file objects after the object that could not be generated
12 TeamConnection exception caused DataAtlas generate to fail

View the `Build Message` for complete details of the build.

Special Considerations

DataAtlas generates ALTER TABLE ADD FOREIGN KEY DDL when generating CREATE TABLE DDL, even if the DDL for the table that is the parent of the foreign key has not been generated yet. For this reason, the Oracle sample build script reorders the ALTER DDL at the end of the DDL output file before exiting.

However, it is more difficult to correctly reorder CREATE VIEW DDL so that it is sequentially correct with respect to the creation of VIEWS referenced by other VIEWS. You might have to manually reorder the generated CREATE VIEW DDL before installing it on your target Oracle database.

You can have multiple Oracle objects in one output part, but you cannot mix Oracle objects with other types of objects. If you do, you get an error while executing the build.

DataAtlas IMS Sample Script Settings

Input Parameters

See “Input Parameters” on page 145 for a general discussion on input parameters.

Return Codes

0 Successful build
8 DataAtlas generate exception; generate attempts to process input file objects after the object that could not be generated
12 TeamConnection exception caused DataAtlas generate to fail

View the `Build Message` for complete details of the build.

Special Considerations

If you connect more than one DBD or PSB to the output file object in your build tree, the build will fail. This also is true if you connect a mixture of DBDs and PSBs.

You should connect only one DBD or PSB per output file object per build tree. You can always generate multiple build trees by connecting them to an output object associated with a NULL builder. For more information about generating multiple build trees, see *TeamConnection User’s Guide.*

DataAtlas COBOL Sample Script Settings

Input Parameters

See “Input Parameters” on page 145 for a general discussion on input parameters.

PREFIX
String appended to the beginning of a data name

Default is NULL

SUFFIX
String appended to the end of a data name

Default is NULL

SLEVEL
Begin numbering highest-level data names with this value

Default is 1

ILEVEL

Increment level numbering by this value

Default is 5

LANG

Do a COBOL generate if the value is COBOL

Default is COBOL

The parameter names are case-sensitive. Their values, on the other hand, are not case-sensitive; in fact, their case will be maintained, since case-sensitivity could be important to an installation's data naming conventions. For example, the 'Parameters' field of the 'Create Builder' window or the 'Create Parts' window could be `PREFIX=prd SUFFIX=test SLEVEL=5 ILEVEL=10`. Not all parameters need to be specified; those that are not specified will use the default value.

Return Codes

0 Successful build
8 DataAtlas generate exception; generate attempts to process input file objects after the object that could not be generated
12 TeamConnection exception caused DataAtlas generate to fail
16 DATATLAS.EXE did not receive a redirected file as input

View the `Build Message` for complete details of the build

Special Considerations

If you connect more than one such object to the output file object, results are unpredictable.

You should connect only one IncludedSourceDef object per output file object per build tree. You can generate multiple build trees by connecting them to an output object associated with a NULL builder. For more information about generating multiple build trees, see *TeamConnection User's Guide.*

DataAtlas PL/I Sample Script Settings

Input Parameters

See "Input Parameters" on page 145 for a general discussion on input parameters.

SLEVEL

Begin numbering highest-level data names with this value

Default is 1

ILEVEL

Increment level numbering by this value

Default is 1

LANG

Do a PL/I generate if the value is PL/I or PLI

Default is COBOL

The parameter names are case-sensitive. Their values, on the other hand, are not case-sensitive; in fact, their case will be maintained, since case-sensitivity could be important to an installation's data naming conventions. For example, the 'Parameters' field of the 'Create Builder' window or the 'Create Parts' window could be `SLEVEL=5 ILEVEL=10`. Not all parameters need to be specified; those that are not specified will use the default value.

Return Codes

0	Successful build
8	DataAtlas generate exception; generate attempts to process input file objects after the object that could not be generated
12	TeamConnection exception caused DataAtlas generate to fail
16	DATATLAS.EXE did not receive a redirected file as input

View the `Build Message` for complete details of the build

Special Considerations

If you connect more than one such object to the output file object, results are unpredictable.

You should connect only one IncludedSourceDef object per output file object per build tree. You can generate multiple build trees by connecting them to an output object associated with a NULL builder. For more information about generating multiple build trees, see *TeamConnection User's Guide.*

Appendix D. TeamConnection Considerations

This appendix contains points to keep in mind about TeamConnection's interaction with DataAtlas.

Renaming Objects

Renaming an object is a special maintenance activity because it occurs in a separate TeamConnection transaction from any other changes made to an object. Because of this, it is possible to make changes to an object, including renaming it, and commit all the changes but fail to change its name. If this occurs, you receive a Store Failure message. If the failure applies to the rename activity, it is likely that the other changes you made to the object have, in fact, succeeded. This becomes obvious if you cancel out of the notebook after receiving the error message and then try to reopen the object. If your other changes are intact, you see them displayed in the notebook.

Another complexity of renaming occurs when multiple users are making changes to the same object. It is possible for two users to search for the same object and bring it into their DataAtlas workfolders. If one user subsequently changes the name of that object, the other user can search again and find the object with the name changed, but the name change will not be reflected in the workfolder until the workfolder is closed and reopened.

Workfolders and the TeamConnection Cache

When an object is opened from a workfolder, it's loaded from the TeamConnection server into a cache in the DataAtlas client machine's local memory. Even after the object's notebook is closed, it remains in the cache, so that it can be reopened quickly without having to access the TeamConnection server. Only when both the notebook and the workfolder are closed is the cached copy of the object removed from local memory. This means that if someone else makes a change to an object after you have reviewed it, reopening the object will still show the unchanged cached copy rather than the updated object on the TeamConnection server. If you close the workfolder and reopen it, opening the object's notebook will load the updated object.

If more than one workfolder is open and one of the workfolders holds a cached copy of an object that is related to an object in another workfolder, closing the first workfolder may not remove the object from the TeamConnection cache, as it can be held in the cache by the relationship. For

example, if workfolder *work1* is open and contains a table *T1* while workfolder *work2* is open and contains table *T1's* table space *TS1*, and if both the table and table space have been opened and closed, then workfolder *work1* and table *T1* will still be in the cache after closing. Only when all notebooks and workfolders for all related objects are closed is it certain that an object is no longer in the TeamConnection cache. You can avoid these complexities by having only one workfolder open at a time, and by closing notebooks whose workfolders you have already closed.

TeamConnection Locking

To make DataAtlas Dictionary and Designer easier to use, all notebooks use an "optimistic" TeamConnection locking policy. To understand this policy, consider what happens when you make changes to a relational table:

1. You double-click the table's icon in the workfolder.
2. DataAtlas goes to the TeamConnection server and loads a copy of the current version of the table into local memory. The table's notebook appears.
3. You make all your changes, click **OK**, enter any check-in remarks, and press click **Store**.
4. DataAtlas locks the table on the server.
5. DataAtlas copies your updated version of the table to the server and unlocks the table.

This process is "optimistic" in that the table isn't locked before you begin to edit it. So if you are just browsing its definition, or if you decide to cancel your changes, DataAtlas doesn't have to explicitly unlock the table. Compare this to a "pessimistic" policy: DataAtlas would have to explicitly lock the table before you made changes and unlock it after you saved them.

The optimistic policy works well because simultaneous editing is rare. In the example, if someone else had made a change to the same table and saved their changes while you were still working, DataAtlas would detect that your copy of the table was "stale" in step 4 and give you a warning. You would have to refresh the table in your TeamConnection cache and re-edit it.

If you choose, you can use TeamConnection to lock DataAtlas objects to prevent any other user from updating them. Each object will remain locked until you store it using DataAtlas or until you unlock it using TeamConnection. Remember that if you decide not to change a locked object, you'll have to explicitly unlock it before anyone else can change it.

Concurrent Versus Serial Development

TeamConnection allows releases to be defined for either *concurrent* or *serial* development. In serial development, DataAtlas users in different work areas are prevented from making changes to the same objects, so one user's changes aren't inadvertently overwritten by another's. If one user changes an object in a work area, a user of another work area can change the same object only by:

- Waiting until the first work area is integrated and then refreshing the second work area, or
- Linking the second work area to the first work area

In concurrent development, both users can make their changes independently of each other, but if the objects changed in the work areas overlap, the changes have to be merged. The integration of the first work area occurs normally. The work area integrated last must be refreshed, which generates collision records for any objects changed in **both** work areas. For file parts, the TeamConnection Merge utility can merge the two versions of each common file, so both users' changes can be retained. DataAtlas objects, however, cannot be merged this way. If a work area refresh results in collision records, the user must choose between these options for each object changed in both work areas:

1. Manually update the second version of the object to include the first version's changes, and then **reject** the collision record (which causes the second version to replace the first when integrated).
2. Leave the first version of the object alone by **accepting** the collision record.

After all the collision records are accepted or rejected, the second work area can be integrated. If any collision records are accepted (leaving the objects unchanged), the user can create a new work area in which to make the remaining changes. Option 1 is preferable because option 2 leaves an interval when some of the new changes are unintegrated.

Only the objects that **overlap** the two work areas are in conflict, but if the overlap is large or complex, another option is to abandon the second work area completely, create a new one based on the other user's integrated changes, and make all the changes again. For example, if the changes in the first work area substantially affect the environment or assumptions under which the second work area is developed, its changes might require a large amount of rework before integration.

With serial development, conflicts are identified sooner and the first-saved changes can't be inadvertently overwritten. With concurrent development, conflicts are identified later and the last-saved changes can overwrite the first-saved. For these reasons, you should use serial development releases with

DataAtlas objects, unless your project requires the independent work supported by concurrent development. If you use concurrent development, you should have good project procedures to manage the complexity.

Appendix E. PL/I Supported Data and Nondata Attributes

The following table summarizes the PL/I supported data attributes. PL/I data attributes are used to describe computational data, program-control data, and program characteristics. DataAtlas supports only those data attributes that can be associated with computational data.

Table 5. PL/I Data Attributes

Attribute	Populate support	SDE Notebook support	IS/SDS Notebook support
Area	no	n/a	no
Binary	yes	yes	yes
Bit	yes	yes	yes
Character	yes	yes	yes
Complex	yes	yes	yes
Decimal	yes	yes	yes
Dimension	yes	n/a	yes
Entry	no	no	no
File	no	no	no
Fixed	yes	yes	yes
Float	yes	yes	yes
Format	no	no	no
Graphic	yes	yes	yes
Handle	no	no	no
Label	no	no	no
Nonvarying (SAA2 only)	yes	yes	yes
Offset	no	n/a	no
Ordinal	no	no	no
Picture	yes	yes	yes
Pointer	yes	no	yes
Precision	yes	yes	yes
Real	yes	yes	yes
Returns	no	no	no
Signed (WS only)	yes	yes	yes
Structure	yes	n/a	yes
Task	no	no	no
Type	no	no	no
Unsigned (WS only)	yes	yes	yes
Union (WS only)	no	n/a	no

Table 5. PL/I Data Attributes (continued)

Attribute	Populate support	SDE Notebook support	IS/SDS Notebook support
Varying	yes	yes	yes
VaryingZ (WS Only)	yes	yes	yes

The following table summarizes PL/I nondata attributes. PL/I nondata attributes are used to describe nondata elements (for example, built-in functions) or provide additional descriptions for elements that have other data attributes. Data Atlas will only support those attributes that provide additional description for elements.

Table 6. PL/I Nondata Attributes

Attribute	Populate support	IS/SDS Notebook support
Abnormal (WS only)	no	no
Aligned	yes	yes
Assignable (WS only)	no	no
Automatic	yes	yes
Based	yes (But only the attribute is supported. The locator-reference for BASED is not populated.)	yes (Your specifications are used when you generate an include file.)
Buffered	no	no
Builtin	no	no
Byaddr	no	no
Byvalue	no	no
Condition	no	no
Connected (WS Only)	no	no
Controlled	yes	yes
Defined	no	yes (An item that DEFINES another item can only define a sibling, the most recent, non-DEFINES sibling under the parent. This follows the same rule as the COBOL for REDEFINES clause. The acceptable value is derived and filled in if one is possible. Your specifications are used when you generate an include file.)
Direct	no	no

Table 6. PL/I Nondata Attributes (continued)

Attribute	Populate support	IS/SDS Notebook support
Environment	no	no
Exclusive	no	no
External	yes	yes
Generic	no	no
Hexadec	no	no
IEEE	no	no
Initial	no	yes (Your specifications are used when you generate an include file.)
Input	no	no
Internal	yes	yes
Keyed	no	no
Like	no	no
List	no	no
Native (WS only)	no	no
Nonassignable (WS only)	no	no
Nonconnected (WS only)	no	no
Nonnative (WS only)	no	no
Normal (WS only)	no	no
Optional (WS only)	no	no
Options	no	no
Output	no	no
Parameter	no	no
Position (w/Defined)	no	no
Print	no	no
Record	no	no
Reserved (WS only)	no	no
Sequential	no	no
Segmented	no	no
Static	yes	yes
Stream	no	no
Unaligned	yes	yes
Unbuffered	no	no
Update	no	no
Value (WS only)	no	no
Variable	no	no

Appendix F. Views and Attributes of Object Types

Table 7 identifies the views (**boldface**) and attributes of object types in the TeamConnection database.

Attributes shown with a dot-dot (..) connection to a view are relationships. For example, under **DAQDataElement**:

`elementAliases..DAQDEAlias`

indicates a connection between the relationship `elementAliases` and the view **DAQDEAlias**. When you use a relationship in a SELECT clause, enter an attribute of the target view in place of the view name. For example:

`elementAliases..elementAlias`

Table 7. Views and Attributes of Object Types

View / Attribute Name	Data Type	Description
Shareable Data Element		
DAQDataElement		
dataElement	character	name of shareable data element
elementDescription	character	description of shareable data element
elementDescType	character	type of description
elementAliases..DAQDEAlias		relationship to data element alias
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
stringLength	integer	maximum length of string data in bits, bytes or double-bytes
stringVaryingCode	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating point, 5-bit string, 6-character string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-MLSLabel (Oracle)
vagSignFormat	boolean	VisualAge Generator sign format
DAQDEAlias		
elementAlias	character	alias or usage name of shareable data element
forDataElement..DAQDataElement		relationship to shareable data element
DAQDataElementUsage		
dataElement	character	name of shareable data element
elementDescription	character	description of shareable data element
elementDescType	character	type of description
columnName	character	name of column in a relational table which uses the data element
inTableDefinition.. DAQTableDefinitionUsage		relationship to table definition which owns the column
imsField	character	name of field in IMS DBD which uses the data element
imsSegment	character	name of segment in IMS DBD which owns the field
inDBD..DAQDBDUsage		relationship to IMS DBD which owns the segment
imsFieldS	character	name of field in IMS DBD which is mapped by a data structure and uses the data element
imsSegmentS	character	name of segment in IMS DBD which owns the field
inDBDs..DAQDBDUsage		relationship to IMS DBD which owns the segment
vagDataItem		name of VisualAge Generator data item which uses the data element
inDataStructure..DAQDSUsage	character	relationship to shareable data structure which uses the data element
inIncludedSource..DAQIncludedSource	character	relationship to included source definition which uses the data element
inDataModel		name of data model which uses the data element
Relational Objects		

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
DAQTableDefinitionUsage		
tableDefinition	character	name of shareable table definition
tableDefDescription	character	description of shareable table definition
tableDefDescType	character	type of description
definesDB2csTable..DAQDB2csTable		relationship to DB2 UDB Table
definesDB2390Table..DAQDB2390Table		relationship to DB2/390 Table
definesOracleTable..DAQOracleTable		relationship to OracleTable
DAQTableDefinition		
tableDefinition	character	name of shareable table definition
tableDefDescription	character	description of shareable table definition
tableDefDescType	character	type of description
inRelationalDesign..DAQRelationalDesign		name of relational design which contains the table definition
columnName	character	name of column in the table definition
columnDesignName	character	name of attribute which defined the column
columnDescription	character	description of column
columnDescriptionType	character	type of description
usesDataElement..DAQDataElement		relationship to shareable data element
localDataTypeCode	integer	data type code: 0-binary number,1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating point, 5-bit string, 6-character string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-MLSLabel (Oracle)
localStringLength	integer	maximum length of string data in bits, bytes or double-bytes
localStringVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
localNumericPrecision	smallint	numeric precision in bits or digits depending on data type
localDecimalScale	smallint	position of binary or decimal point
localAsBitData	boolean	flag indicating character string should be treated as bit (binary) data
localIsLOB	boolean	flag indicating string is a large object (LOB)
definedByColumnName	character	name of primary key column which defines the foreign key column
inSourceTableDefinition.. DAQTableDefinition		table definition containing primary key which defines the foreign key column

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
allowNulls	character	indicates whether null values are allowed (Y or N)
defaultOption	character	Y-system default,I-initialValue,N-null,C-current SQLID,U-User ID,D-current date,T-Current time,S-current timestamp,K-castFunction and value in initialValue,O-castFunction and current timestamp,P-castFunction and current time,Q-castFunction and current date
initialValue	character	initial value for the column
castFunctionName	character	name of cast function used to set default column value
columnComment	character	SQL comment on the column
columnLabel	character	SQL label for the column
fieldProc	character	name of field procedure for the column (DB2/390 only)
constants	character	list of constants needed for field procedure (DB2/390 only)
averageLength	smallint	average length of a VARCHAR or LONG VARCHAR column (DB2/390 only)
initialNoOfDistinctValues	integer	initial number of distinct values of the column (DB2/390 only)
arithmeticFunctionsFrequency	smallint	how frequently SQL statements containing arithmetic functions reference the column (DB2/390 only)
joinFrequency	smallint	how frequently SQL join statements use the column (DB2/390 only)
rangePredicatesFrequency	smallint	how frequently SQL statements containing <, >, <=, >=, LIKE, BETWEEN reference the column (DB2/390 only)
skewingFactor	smallint	number of most frequently occurring values (DB2/390 only)
skewingPercentage	smallint	percentage of all rows in which the most frequently occurring values appear (DB2/390 only)
sortingFunctionFrequency	smallint	how frequently SQL statements containing GROUP BY, ORDER BY, DISTINCT reference the column (DB2/390 only)
updateFrequency	smallint	how frequently the column is updated (DB2/390 only)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
whereClauseFrequency	smallint	how frequently SQL WHERE statements reference the column (DB2/390 only)
loggedLOB	boolean	whether changes to the LOB column are logged (DB2 UDB only)
compactLOB	boolean	whether data in the LOB column should be compact (DB2 UDB only)
primaryFlag	character	whether the column participates in the primary key (Y or N) (DB2 UDB only)
auditOption	character	type of auditing on table: N-none, C-changes, A-all
tableComment	character	SQL comment on the table
tableLabel	character	SQL label for the table
hasKeyDefinition..DAQKeyDefinition		relationship to definition of primary, foreign, or unique key
primaryKeyFlag	boolean	indicates a primary key
foreignKeyFlag	boolean	indicates a foreign key
uniqueFlag	character	indicates whether key columns are unique (Y or N)
pkConstraintName	character	primary key constraint name (DB2 UDB only)
pkConstraintComment	character	SQL comment on the primary key constraint
hasPrimaryKey..DAQKeyDefinition		relationship to definition of key columns of the primary key
checkConstraintName	character	name of check constraint on the table
checkConstraintComment	character	SQL comment on check constraint
checkConstraintCondition	character	check constraint condition clause
definesTable	character	name of DB2/390, DB2 UDB or Oracle table defined by the table definition
entityName	character	name of the entity transformed into the table
dataModel	character	name of the dataModel which contains the entity
DAQKeyDefinition		
keyDefinition	character	name of the definition of columns forming a key
keyColumnName	character	name of a column in the key
ascDescFlag	character	indicates whether the column is sorted ascending (A) or descending (D) when the key definition is used by an index
inTableDefinition..DAQTableDefinition		relationship to table definition which contains the column
foreignKeyConstraintName	character	constraint name of the foreign key

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
fkConstraintComment	character	SQL comment on the foreign key constraint
foreignKeyDeleteOption	character	delete option of the foreign key: C-cascade, R-restrict, N-set null, A-no action
parentKeyDefinition	character	name of the key definition of the parent table
parentTableDef..DAQTableDefinition		relationship to the table definition of the parent table
parentKeyIsPrimary	boolean	whether parent key is a primary key
parentKeyIsUnique	character	whether the primary key is a unique key (Y or N)
foreignKeyDependentTable	character	name of DB2/390, DB2 UDB or Oracle table which contains the foreign key
foreignKeyParentTable	character	name of DB2/390, DB2 UDB or Oracle table which is a parent table of the foreign key
dependentConstraintName	character	constraint name of a foreign key using the primary or unique key
dependentTableDef..DAQTableDefinition		relationship to the table definition of the dependent table
primaryKeyParentTable	character	name of DB2/390, DB2 UDB or Oracle table which contains the primary key
primaryKeyDependentTable	character	name of DB2/390, DB2 UDB or Oracle table which is dependent upon the primary key
usedByIndex	character	name of DB2/390, DB2 UDB or Oracle index which uses the key definition
relationshipName	character	name of the relationship transformed into the foreign key
dataModel	character	name of the dataModel which contains the relationship
DAQViewDefintion		
viewDefinition	character	name of shareable view definition
viewDefDescription	character	description of shareable view definition
viewDefDescType	character	type of description
inRelationalDesign..DAQRelationalDesign		name of relational design which contains the view definition
columnName	character	name of a column in the view definition
columnComment	character	SQL comment on the column
viewComment	character	SQL comment on the view
viewLabel	character	SQL label of the view

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
checkOption	character	whether the DBMS should check the results of updates to the view against the view's search condition. Valid values are: Y - use local check, C - use CASCADED check, N - no check
DAQRelationalDesign		
relationalDesign	character	name of the relational design
relDesignDescription	character	description of relational design
relDesignDescType	character	type of description
containsTableDef..DAQTableDefinition		relationship to table definition
containsViewDef..DAQViewDefinition		relationship to view definition
hasDB2390PhysicalDesign.. DAQDB2390PhysicalDesign		relationship to DB2/390 physical design
hasDB2csPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to DB2 UDB physical design
hasOraclePhysicalDesign.. DAQOraclePhysicalDesign		relationship to Oracle physical design
dataModel	character	data model which represents the relational design
DAQDB2390PhysicalDesign		
db2390PhysicalDesign	character	name of DB2/390 physical design
physDesignDescription	character	description of DB2/390 physical design
physDesignDescType	character	type of description
inRelationalDesign..DAQRelationalDesign		relationship to relational design which contains the physical design
aliasOrSynonym..DAQDB2390AliasSynonym		relationship to a DB2/390 alias or synonym in the physical design
db2390Table..DAQDB2390Table		relationship to a DB2/390 table in the physical design
db2390Tablespace..DAQDB2390Tablespace		relationship to a DB2/390 table space in the physical design
db2390Storagegroup.. DAQDB2390Storagegroup		relationship to a DB2/390 storage group in the physical design
db2390View..DAQDB2390View		relationship to a DB2/390 view in the physical design
db2390Index..DAQDB2390Index		relationship to a DB2/390 index in the physical design
db2390Database..DAQDB2390Database		relationship to a DB2/390 database in the physical design

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
db2390Bufferpool..DAQDB2390Bufferpool		relationship to a DB2/390 buffer pool in the physical design
DAQDB2csPhysicalDesign		
db2csPhysicalDesign	character	name of DB2 UDB physical design
physDesignDescription	character	description of DB2 UDB physical design
physDesignDescTyp	character	type of description
inRelationalDesign..DAQRelationalDesign		relationship to relational design which contains the physical design
db2csTable..DAQDB2csTable		relationship to a DB2 UDB table in the physical design
db2csTablespace..DAQDB2csTablespace		relationship to a DB2 UDB table space in the physical design
db2csView..DAQDB2csView		relationship to a DB2 UDB view in the physical design
db2csIndex..DAQDB2csIndex		relationship to a DB2 UDB index in the physical design
db2csDatabase..DAQDB2csDatabase		relationship to a DB2 UDB database in the physical design
DAQOraclePhysicalDesign		
oraclePhysicalDesign	character	name of Oracle physical design
physDesignDescription	character	description of Oracle physical design
physDesignDescTyp	character	type of description
inRelationalDesign..DAQRelationalDesign		relationship to relational design which contains the physical design
oracleTable..DAQOracleTable		relationship to a Oracle table in the physical design
oracleTablespace..DAQOracleTablespace		relationship to a Oracle table space in the physical design
oracleView..DAQOracleView		relationship to a Oracle view in the physical design
oracleIndex..DAQOracleIndex		relationship to a Oracle index in the physical design
DAQRelationalSystem		
relationalSystem	character	name of the relational system
rdbProductName	character	name of the relational database product
rdbProductVersion	character	version of the relational database product
sysplex	boolean	identifies the DB2 for MVS system as in a data sharing environment
systemDescription	character	description of the relational system
systemDescType	character	type of description

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
creatorID	character	name of a creator or schema in the relational system
db2csTable..DAQDB2csTable		relationship to a DB2 UDB table in the system
db2csView..DAQDB2csView		relationship to a DB2 UDB view in the system
db2csIndex..DAQDB2csIndex		relationship to a DB2 UDB index in the system
oracleTable..DAQOracleTable		relationship to a Oracle table in the system
oracleView..DAQOracleView		relationship to a Oracle view in the system
oracleIndex..DAQOracleIndex		relationship to a Oracle index in the system
db2390Table..DAQDB2390Table		relationship to a DB2/390 table in the system
db2390View..DAQDB2390View		relationship to a DB2/390 view in the system
db2390AliasSynonym.. DAQDB2390AliasSynonym		relationship to a DB2/390 alias or synonym in the system
db2390Index..DAQDB2390Index		relationship to a DB2/390 index in the system
db2390Indexspace.. DAQDB2390Indexspace		relationship to a DB2/390 index space in the system
mvsICFCatalogIS..DAQICFCatalog		relationship to a ICF Catalog used by a index space in the system
mvsICFCatalogISP..DAQICFCatalog		relationship to a ICF Catalog used by a index space partition in the system
db2390Tablespace.. DAQDB2390Tablespace		relationship to a DB2/390 table space in the system
mvsICFCatalogTS..DAQICFCatalog		relationship to a ICF Catalog used by a table space in the system
mvsICFCatalogTSP..DAQICFCatalog		relationship to a ICF Catalog used by a table space partition in the system
db2390Database..DAQDB2390Database		relationship to a DB2/390 database in the system
db2390StorageGroup.. DAQDB2390Storagegroup		relationship to a DB2/390 storage group in the system
mvsICFCatalogSG..DAQICFCatalog		relationship to a ICF Catalog used by a storage group in the system
mvsVolume..DAQMVSVolume		relationship to a MVS volume used by a storage group in the system
db2390Bufferpool..DAQDB2390Bufferpool		relationship to a DB2/390 buffer pool in the system
oracleTablespace..DAQOracleTablespace		relationship to a Oracle table space in the system
db2csTablespace..DAQDB2csTablespace		relationship to a DB2 UDB table space in the system

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
db2csDatabase..DAQDB2csDatabase		relationship to a DB2 UDB database in the system
DAQDB2390Table		
db2390Table	character	name of DB2/390 table
tableDescription	character	description of table
tableDescType	character	type of description
creatorID	character	authorization ID of the creator of the table
inRelationalSystem.. DAQRelationalSystem		relationship to the relational system which owns the table
usesTableDefinition..DAQTableDefinition		relationship to the table definition which defines the table
inDatabase..DAQDB2390Database		relationship to the database which contains the table
inTablespace..DAQDB2390Tablespace		relationship to the table space which contains the table
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the table
hasIndex..DAQDB2390Index		relationship to the indexes defined on the table
hasAliasOrSynonym.. DAQDB2390AliasSynonym		relationship to alias or synonym of the table
clusteringIndexName	character	name of the clustering index of the table
cardinality	integer	total number of rows in the table from last time statistics were gathered from DB2
catalogOBID	smallint	an internal identifier of the table
ccsid	character	value to specify in the CCSID clause: E - EBCDIC or A - ASCII
nPages	integer	the total number of pages on which rows of the table appear the last time statistics were gathered from DB2
dataCaptureFlag	character	value to specify in the DATACAPTURE clause: N - None C - Changes
editProc	character	name of the edit proc associated with the table
pctPages	smallint	percentage of active table space pages that contain rows of the table the last time statistics were gathered from DB2
restrictDrop	boolean	whether or not the RESTRICT ON DROP clause is to be generated for the table

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
pctRowComp	smallint	percentage of rows, multiplied by 100, compressed within the total number of active rows in the table the last time statistics were gathered from DB2
recLength	smallint	maximum length of any record in the table the last time statistics were gathered from DB2
statsTime	character	the date and time when the last invocation of DB2 RUNSTATS updated the catalog statistics. The timestamp has a maximum of 26 digits and separators formatted as: yyyy-mm-dd-hh.mm.ss.nnnnnn.. Where: yyyy - year, mm - month, dd - day, hh - hour, mm - minute, ss second, nnnnnn - microsecond
catalogStatus	character	status of the table when the latest statistics were gathered from DB2: I - tables's definition is incomplete because it lacks a primary index X - table has a primary index blank - table has no primary key, or is a catalog table
validProc	character	name of the valid proc associated with the table
randomUpdate	boolean	whether the rows in the table will be updated randomly or sequentially
numPartitions	smallint	number of table partitions. A table partition is specified as part of the physical design and is the basis for the proposal of a table space partition
partitionNumber	smallint	table partitionidentifier
partitionColumnName	character	name of a column in the partitioning index
confidenceFactor	smallint	confidence factor of the growth and delete rate values
deletePeriod	character	period of the delete rate D - daily W - weekly M - monthly Q - quarterly
deleteRate	smallint	relative rate of DELETE operations on the table (1-10)
growthPeriod	character	period of the growth rate. D - daily W - weekly M - monthly Q - quarterly
growthRate	smallint	relative rate of increase in the number of rows in the table (1-10)
initialNoOfRows	integer	predicted initial number of rows in the table
maintenancePeriod	smallint	amount of time for which the growth and delete rates are calculated

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
alterTableFrequency	smallint	how frequently an alter table is executed on the table
deleteFrequency	smallint	how frequently a delete statement is executed on the table
insertFrequency	smallint	how frequently an insert statement is executed on the table
reorgFrequency	smallint	how frequently an reorg is executed on the table
selectFrequency	smallint	how frequently a select statement is executed on the table
unloadReloadFrequency	smallint	how frequently an unload and reload operation is executed on the table
updateFrequency	smallint	how frequently an update statement is executed on the table
concurrency	character	concurrency requirements. L - low M - medium H - high
securityOptionSet	boolean	whether the data is security sensitive (the basis of the proposing the ERASE option)
DAQDB2390View		
db2390View	character	name of DB2/390 view
viewDescription	character	description of view
viewDescType	character	type of description
creatorID	character	authorization ID of the creator of the view
inRelationalSystem.. DAQRelationalSystem		relationship to the relational system which owns the view
usesViewDefinition..DAQViewDefinition		relationship to the view definition which defines the view
select	character	select statement using actual table or view names
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the view
hasAliasOrSynonym.. DAQDB2390AliasSynonym		relationship to alias or synonym of the view
DAQDB2390AliasSynonym		
db2390AliasOrSynonym	character	name of DB2/390 alias or synonym
aliasSynonymDescription	character	description of alias or synonym
aliasSynonymDescType	character	type of description
synonymAliasFlag	character	indicates whether the is a synonym (S) or alias (A)
creatorID	character	authorization ID of the alias or synonym

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
inRelationalSystem.. DAQRelationalSystem		relationship to the relational system which owns the alias or synonym
comment	character	SQL comment on the alias or synonym
label	character	SQL label for the alias or synonym
inDB2390PhysicalDesign.. DAQDB2390PhysicalDesign		relationship to DB2/390 physical design which contains the alias/synonym
forDB2390Alias.. DAQDB2390AliasSynonym		relationship to DB2/390 alias or synonym which this is alias of
forDB2390Table..DAQDB2390Table		relationship to DB2/390 table which this is alias of
forDB2390View..DAQDB2390View		relationship to DB2/390 view which this is alias of
hasAliasOrSynonym.. DAQDB2390AliasSynonym		relationship to a DB2/390 alias or synonym of this alias
DAQDB2390Index		
db2390Index	character	name of DB2/390 index
indexDescription	character	description of index
indexDescType	character	type of description
creatorID	character	authorization ID of the creator of the view
inRelationalSystem.. DAQRelationalSystem		relationship to the relational system which owns the view
usesKeyDefinition..DAQKeyDefinition		relationship to definition of columns in the index
forTable..DAQDB2390Table		relationship to DB2/390 table on which index is created
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to DB2/390 physical design which contains the index
isClusterIndexFor..DAQDB2390Table		relationship to DB2/390 table which is clustered by the index
partitionsTablespace.. DAQDB2390Tablespace		relationship to DB2/390 table space which is partitioned by the index
indexType	character	identifies the index as Type 1 or Type 2
uniqueFlag	character	Y - unique N - not unique
whereNotNull	boolean	whether the WHERE NOT NULL clause is specified for the Index
pieceSize	integer	value in Kilobytes for the PIECESIZE clause for the index
clusterFlag	character	whether CLUSTER is specified for the index (Y/ N)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
clusterRatio	integer	percentage of rows that are in clustering order the last time statistics were gathered from DB2
firstKeyCard	integer	number of distinct values of the first key column the last time statistics were gathered from DB2
fullKeyCard	integer	number of distinct values of the key the last time statistics were gathered from DB2
numLeafs	integer	number of active leaf pages in the index the last time statistics were gathered from DB2
numLevels	integer	number of levels in the index tree the last time statistics were gathered from DB2
space	integer	number of kilobytes of DASD storage allocated to the index the last time statistics were gathered from DB2
statsTime	character	date and time when the last invocation of RUNSTATS updated the statistics. The timestamp has a maximum of 26 digits and separators formatted as: yyyy-mm-dd-hh.mm.ss.nnnnnn.. Where: yyyy - year, mm - month, dd - day, hh - hour, mm - minute, ss - second, nnnnnn - microsecond
numRows	integer	predicted number of rows represented in the index
numDistinctValues	integer	predicted number of distinct values in the index
inIndexspace..DAQDB2390Indexspace		relationship to the index space containing the index
DAQDB2390Indexspace		
db2390Indexspace	character	name of the index which uses the index space
closeRule	character	close rule option for an index Y - yes: data set is eligible for closing. N - no: data set is not eligible for closing
dsetPasswordFlag	character	whether the data sets of index space are password protected (Y / N)
dsetPassword	character	password for the data sets of the index
deferRule	character	whether the index is built during execution of the CREATE INDEX statement N - The index is built (this is the default.) Y - The index is not built
subPages	smallint	number of subpages for each physical page

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
idxFreePage	smallint	number of pages that are loaded before a page is left as free space
idxPercentFree	smallint	percentage of each subpage or nonleaf page that is left as free space
idxGBPCache	character	whether or not the GBPCACHE block is to be generated for the index and the value to be set. If this attribute is null, no GBPCACHE will be generated. Other values are 'C' - changed or 'A' - all
usesBufferpool..DAQDB2390Bufferpool		relationship to buffer pool used by the index
idxICFCatalog..DAQICFCatalog	relationship to ICF catalog used by the index	
idxStorageGroup.. DAQDB2390Storagegroup		relationship to the storage group used by the index
idxEraseRule	character	action to be performed when the index is dropped: Y - erase N - do not erase
idxPrimaryQty	integer	primary space allocation for DB2 defined data sets in kilobytes
idxSecondaryQty	integer	secondary space allocation for DB2 defined data sets in kilobytes
numPartitions	smallint	number of partitions
idxPartNumber	smallint	the partition number. when zero statistics apply to entire index space
idxPartHighKeyValues	character	value for each partition. These values are concatenated, and the concatenation of all the values is the highest value of the key in the corresponding partition of the index
idxPartFreePages	smallint	number of pages that are loaded before a page is left as free space
idxPartPctFree	smallint	percentage of each subpage or nonleaf page that is left as free space
idxPartGBPCache	character	whether or not the GBPCACHE block is to be generated for this index and the value to be set. If this attribute is null, no GBPCACHE will be generated. Other values are 'C' - changed or 'A' - all
idxPartStogroup.. DAQDB2390Storagegroup		relationship to the storage group used by the index partition

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
idxPartEraseRule	character	action to be performed when the index is dropped: Y - erase N - do not erase
idxPartPrimaryQty	integer	primary space allocation for DB2 defined data sets in kilobytes
idxPartSecondaryQty	integer	secondary space allocation for DB2 defined data sets in kilobytes
idxPartICFCatalog..DAQICFCatalog		relationship to ICF catalog used by the index partition
idxPartCardinality	integer	the number of rows referred to by index partition
idxPartSpace	integer	number of kilobytes of DASD storage allocated to the index space partition
idxPartFarOffOpt	integer	number of referred to rows far from optimal position because of an insert into a full page
idxPartNearOpt	integer	number of referred to rows near, but not at optimal position because of an insert into a full page
idxPartLeafDis	integer	100 times the average number of pages between successive leaf pages of the index
idxPartStatsTime	character	date and time when the last invocation of RUNSTATS updated the statistics. The timestamp has a maximum of 26 digits and separators formatted as: yyyy-mm-dd-hh.mm.ss.nnnnnn.. Where: yyyy - year, mm - month, dd - day, hh - hour, mm - minute, ss - second, nnnnnn - microseconds
DAQDB2390Tablespace		
db2390Tablespace	character	name of DB2/390 table space
tablespaceDescription	character	description of table space
tablespaceDescType	character	type of description
inRelationalSystem.. DAQRelationalSystem		relationship to the relational system which owns the table space
inDatabase..DAQDB2390Database		relationship to the database which contains the table space
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the table space
containsTable..DAQDB2390Table		relationship to the tables in the table space
partitionedByIndex..DAQDB2390Index		relationship to the index that is the partitioning index of the table space
usesBufferpool..DAQDB2390Bufferpool		relationship to the buffer pool used by the table space

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
lockSize	character	locksize for the table space: A - any, P - page, R- row, S - table space, T - table
lockmax	character	whether or not the LOCKMAX clause is to be generated for the table space and the value to be set. If this attribute is null, no LOCKMAX will be generated. Other values are 'S' - system or 'V' - value specified in "maxValue"
maxValue	integer	the user-defined value for the LOCKMAX keyword
lockPart	boolean	whether or not the LOCKPART clause will be generated
maxRows	smallint	maximum number of rows DB2 will put on a page (1 to 255)
closeRule	character	close rule option for the table space Y - yes: data set is eligible for closing. N - no: data set is not eligible for closing
ccsid	character	whether or not the CCSID clause is to be generated for the table space and the value to be set. If this attribute is null, no CCSID will be generated. Other values are E - EBCDIC or A - ASCII
largeTS	boolean	whether or not the LARGE keyword will be generated on a CREATE TABLESPACE statement
dsetPasswordFlag	character	whether the data sets of table space are password protected (Y/N)
dsetPassword	character	password for the data sets of the table space
compressFlag	character	whether data compression applies to the rows of the table space. Y - yes N - no
tsFreePage	smallint	number of pages that are loaded before a page is left as free space
tsPercentFree	smallint	percentage of each subpage or nonleaf page that is left as free space
tsGBPCache	character	whether or not the GBPCACHE block is to be generated for the table space and the value to be set. If this attribute is null, no GBPCACHE will be generated. Other values are 'C' - changed or 'A' - all
nActivePages	integer	number of active pages in the table space
space	integer	number of kilobytes of DASD storage allocated to the table space

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
usageIntent	character	applications the database will be used for: Q - QMF B - batch T - test R - random
segSize	smallint	the size of each segment in a segmented table space (must be 4 <= segsize <= 64)
usesStorageGroup.. DAQDB2390Storagegroup		relationship to the storage group used by the table space
tsEraseRule	character	action to be performed when the table space is dropped: Y - erase N - do not erase
tsPrimaryQty	integer	primary space allocation for DB2 defined data sets in kilobytes
tsSecondaryQty	integer	secondary space allocation for DB2 defined data sets in kilobytes
tsICFCatalog..DAQICFCatalog		relationship to ICF catalog used by the table space
numPartitions	smallint	number of partitions
tsPartNumber	smallint	the partition number. when zero statistics apply to entire table space
tsPartFreePage	smallint	number of pages that are loaded before a page is left as free space
tsPartPctFree	smallint	percentage of each subpage or nonleaf page that is left as free space
tsPartGBPCache	character	whether or not the GBPCACHE block is to be generated for this table space and the value to be set. If this attribute is null, no GBPCACHE will be generated. Other values are 'C' - changed or 'A' - all
tsPartStogroup.. DAQDB2390Storagegroup		relationship to the storage group used by the table space partition
tsPartEraseRule	character	action to be performed when the table space is dropped: Y - erase N - do not erase
tsPartPrimaryQty	integer	primary space allocation for DB2 defined data sets in kilobytes
tsPartSecondaryQty	integer	secondary space allocation for DB2 defined data sets in kilobytes
tsPartICFCatalog..DAQICFCatalog		relationship to ICF catalog used by the table space partition
tsPartCardinality	integer	the number of rows referred to by table space partition
tsPartSpace	integer	number of kilobytes of DASD storage allocated to the table space partition

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
tsPartFarOffOpt	integer	number of referred to rows far from optimal position because of an insert into a full page
tsPartNearOpt	integer	number of referred to rows near, but not at optimal position because of an insert into a full page
tsPartPctActive	smallint	percentage of space occupied by rows of data from active tables
tsPartPctDropped	smallint	percentage of space occupied by rows of dropped tables
tsPartPageSave	smallint	percentage of pages, multiplied by 100, saved in the table space or partition as a result of using data compression
tsPartStatsTime	character	date and time when the last invocation of RUNSTATS updated the statistics. The timestamp has a maximum of 26 digits and separators formatted as: yyyy-mm-dd-hh.mm.ss.nnnnnn.. Where: yyyy - year, mm - month, dd - day, hh - hour, mm - minute, ss - second, nnnnnn - microseconds
DAQDB2390Database		
db2390Database	character	name of DB2/390 database
databaseDescription	character	description of database
databaseDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the database
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the database
containsIndex..DAQDB2390Index		relationship to the indexes in the database
containsTablespace.. DAQDB2390Tablespace		relationship to the table spaces in the database
containsTable..DAQDB2390Table		relationship to the tables in the database
ccsid	character	whether or not the CCSID clause is to be generated for the database and the value to be set. If this attribute is null, no CCSID will be generated. Other values are E - EBCDIC or A - ASCII
roShareFlag	character	how the database will be using shared read-only data. O - owner R - read only
usesBufferpool..DAQDB2390Bufferpool		relationship to buffer pool used by the database

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
usesStoragegroup.. DAQDB2390Storagegroup		relationship to the storage group used by the database
usageIntent	character	applications the database will be used for: Q - QMF N - others
DAQDB2390Storagegroup		
db2390Storagegroup	character	name of DB2/390 storage group
storagegroupDescription	character	description of storage group
storagegroupDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the storage group
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the storage group
usedByDatabase..DAQDB2390Database		relationship to databases which use the storage group
usedByTablespace.. DAQDB2390Tablespace		relationship to the table spaces which use the storage group
usedByPartTablespace.. DAQDB2390Tablespace		relationship to partitioned table spaces which use the storage group
tsPartNumber	smallint	partition number of the table space partition which uses the storage group
usedByIndex..DAQDB2390Index		relationship to the indexes which use the storage group
usedByPartIndex..DAQDB2390Index		relationship to partitioned indexes which use the storage group
idxPartNumber	smallint	partition number of the index partition which uses the storage group
passwordFlag	character	whether the ICF catalog is password protected (Y/N)
password	character	VSAM control or master level password of the ICF catalog
usesICFCatalog..DAQICFCatalog		relationship to the ICF catalog used by the storage group
usesVolume..DAQMVSVolume		relationship to the volumes of the storage group
requiredSpace	smallint	space required as calculated by the physical database design function
usageIntent	character	whether storage group should be reserved for table spaces (T) or indexes (I)
space	integer	number of kilobytes of DASD storage allocated to the storage group as determined by the last execution of the STOSPACE utility

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
DAQDB2390Bufferpool		
db2390Bufferpool	character	name of DB2/390 buffer pool
bufferpoolDescription	character	description of buffer pool
bufferpoolDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the buffer pool
inPhysicalDesign.. DAQDB2390PhysicalDesign		relationship to the physical designs which contain the buffer pool
usedByDatabase..DAQDB2390Database		relationship to databases which use the buffer pool
usedByTablespace.. DAQDB2390Tablespace		relationship to table spaces which use the buffer pool
usedByIndex..DAQDB2390Index		relationship to indexes which use the buffer pool
bufferPoolNo	character	buffer pool number
pageSize	character	buffer pool page size: A - 4K page size B - 32K page size
usageIntent	character	
DAQMVSVolume		
MVSVolume	character	name of the MVS volume
volumeID	character	volume ID which identifies volume serial number of an OS/VS storage volume
usedByStoragegroup.. DAQDB2390Storagegroup		relationship to storage groups which use the volume
DAQICFCatalog		
ICFCatalog	character	name of the ICF Catalog
usedByStoragegroup.. DAQDB2390Storagegroup		relationship to storage groups which use the ICF catalog
usedByTablespace.. DAQDB2390Tablespace		relationship to table spaces which use the ICF catalog
usedByPartTablespace.. DAQDB2390Tablespace		relationship to partitioned table spaces which use the ICF catalog
tsPartNumber	smallint	partition number of the table space partition which uses the ICF catalog
usedByIndex..DAQDB2390Index		relationship to indexes which use the ICF catalog
usedByPartIndex..DAQDB2390Index		relationship to partitioned indexes which use the ICF catalog
idxPartNumber	smallint	partition number of the index partition which uses the ICF catalog

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
DAQDB2CSTable		
db2csTable	character	name of DB2 UDB table
tableDescription	character	description of the DB2 UDB table
tableDescType	character	type of description
schema	character	schema which owns the DB2 UDB table
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the table
usesTableDefinition..DAQTableDefinition		relationship to the table definition which defines the table
inPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to the physical designs which contain the table
inDatabase..DAQDB2csDatabase		relationship to the database which contains the table
inPrimaryTablespace.. DAQDB2csTablespace		relationship to the table space that contains the table
usesIndexTablespace.. DAQDB2csTablespace		relationship to the table space which contains indexes on the table
usesLongTablespace.. DAQDB2csTablespace		relationship to the table space which contains LOB data in the table
hasIndex..DAQDB2csIndex		relationship to the indexes defined on the table
DAQDB2CSIndex		
db2csIndex	character	name of DB2 UDB index
indexDescription	character	description of the DB2 UDB index
indexDescType	character	type of description
schema	character	schema which owns the DB2 UDB index
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the index
usesKeyDefinition..DAQKeyDefinition		relationship to the key definition which defines the table
inPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to the physical designs which contain the index
inDatabase..DAQDB2csDatabase		relationship to the database which contains the index
forTable..DAQDB2csTable		relationship to the table on which the index is created
comment	character	SQL comment on the index
uniqueFlag	character	Y - unique N - not unique
DAQDB2CSView		
db2csView	character	name of DB2 UDB view

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
viewDescription	character	description of the DB2 UDB view
viewDescType	character	type of description
schema	character	schema which owns the DB2 UDB view
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the view
usesViewDefinition..DAQViewDefinition		relationship to the view definition which defines the view
select	character	select statement using actual table or view names
inPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to the physical designs which contain the view
inDatabase..DAQDB2csDatabase		relationship to the database which contains the view
DAQDB2CSDatabase		
db2csDatabase	character	name of DB2 UDB database
databaseDescription	character	description of the DB2 UDB database
databaseDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the view
inPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to the physical designs which contain the view
containsTable..DAQDB2csTable		relationship to the tables in the database
containsView..DAQDB2csView		relationship to the views in the database
containsIndex..DAQDB2csIndex		relationship to the indexes in the database
DAQDB2CSTablespace		
db2csTablespace	character	name of DB2 UDB table space
tablespaceDescription	character	description of the DB2 UDB table space
tablespaceDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the view
inPhysicalDesign.. DAQDB2csPhysicalDesign		relationship to the physical designs which contain the view
tsType	character	type of table space: R - regular L - long (can hold LOB data) T - temporary (only for temporary tables)
containsTable..DAQDB2csTable		relationship to tables in the table space
containsIndexForTable..DAQDB2csTable		relationship to tables for which indexes are in the table space
containsLOBForTable..DAQDB2csTable		relationship to tables for which large object (LOB) columns are in the table space

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
DAQOracleTable		
oracleTable	character	name of Oracle table
tableDescription	character	description of the Oracle table
tableDescType	character	type of description
schema	character	schema which owns the Oracle table
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the table
usesTableDefinition..DAQTableDefinition		relationship to the table definition which defines the table
inPhysicalDesign.. DAQOraclePhysicalDesign		relationship to the physical designs which contain the table
inTablespace.. DAQOracleTablespace		relationship to the table space which contains the table
hasIndex..DAQOracleIndex		relationship to the indexes defined on the table
DAQOracleIndex		
oracleIndex	character	name of Oracle index
indexDescription	character	description of the Oracle index
indexDescType	character	type of description
schema	character	schema which owns the DB2 UDB index
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the index
usesKeyDefinition..DAQKeyDefinition		relationship to the key definition which defines the table
inPhysicalDesign.. DAQOraclePhysicalDesign		relationship to the physical designs which contain the index
inDatabase..DAQOracleDatabase		relationship to the database which contains the index
forTable..DAQOracleTable		relationship to the table on which the index is created
comment	character	SQL comment on the index
uniqueFlag	character	Y - unique N - not unique
DAQOracleView		
oracleView	character	name of Oracle view
viewDescription	character	description of the Oracle view
viewDescType	character	type of description
schema	character	schema which owns the Oracle view
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the view

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
usesViewDefinition..DAQViewDefinition		relationship to the view definition which defines the view
select	character	select statement using actual table or view names
inPhysicalDesign.. DAQOraclePhysicalDesign		relationship to the physical designs which contain the view
force	boolean	whether or not the FORCE clause is to be generated
withOption	character	whether or not the WITH clause is to be generated for the view and the value to be set. If this attribute is null, no clause will be generated. Other values are 'RO' - read only or 'CO' - check option
constraintName	character	constraint name that is part of the WITH CHECK OPTION clause
DAQOracleTablespace		
oracleTablespace	character	name of Oracle table space
tablespaceDescription	character	description of the Oracle space
tablespaceDescType	character	type of description
inRelationalSystem..DAQRelationalSystem		relationship to the relational system which owns the view
containsTable..DAQOracleTable		relationship to tables in the table space
containsIndex..DAQOracleIndex		relationship to indexes in the table space
inPhysicalDesign.. DAQOraclePhysicalDesign		relationship to the physical designs which contain the view
IMS Objects		
DAQgsamDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be GSAM for this view
osAccess	boolean	Operating system access method - TRUE for VSAM, false for BSAM
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO. Value is ignored when osAccess is FALSE.
versionString	character	value for VERSION= keyword
ddName1	character	DDname for input file (DD1= keyword)
ddName2	character	DDname for output file (DD2= keyword)
recordFormat	character	File record format (RECFM= keyword). Valid values are ″F″, ″V″, ″FB″, ″VB″, ″U″.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
inputRecordLength	integer	Maximum or fixed record length (first parameter for RECLEN= keyword)
outputRecordLength	integer	Minimum record length (second parameter for RECLEN= keyword)
size1	integer	Record size (SIZE= keyword)
primaryBlockingFactor	integer	Blocking factor (BLOCK= keyword)
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD.
dbdLabel	character	type of description
dbdDescription	character	description of DBD
DAQmsdbDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be MSDB for this view
msdbType	character	Type of MSDB - ″NO″, ″FIXED″, ″TERM″ or ″DYNAMIC″
msdbField	character	Field name for DYNAMIC MSDB
versionString	character	value for VERSION= keyword
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
segmentName	character	name of segment (NAME= keyword).
maximumLength	integer	length of segment (BYTES= keyword).
frequency	character	segment frequency (FREQ= keyword).
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
fieldName	character	name of field (NAME= keyword).

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
DAQhsamDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be HSAM, SHSAM or SHISAM for this view
versionString	character	value for VERSION= keyword
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO.
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescriptioncharacter	description of DBD	
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
ddName1	character	DDname for input file (DD1= keyword)
ddName2	character	DDname for output file (DD2= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
inputRecordLength	integer	Record length of input file (first parameter for RECLEN= keyword)
outputRecordLength	integer	Record length of output file (second parameter for RECLEN= keyword)
size1	integer	Input record size (first parameter for SIZE= keyword)
size2	integer	Output record size (second parameter for SIZE= keyword)
primaryBlockingFactor	integer	Input file blocking factor (first parameter for BLOCK= keyword)
overflowBlockingFactor	integer	Output file blocking factor (second parameter for BLOCK= keyword)
captureRoutine	character	name of data capture exit (for EXIT= keyword)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
exitData	boolean	flag for DATA or NODATA
exitPath	boolean	flag for PATH or NOPATH
exitKey	boolean	flag for KEY or NOKEY
cascade	boolean	flag for CASCADE or NOCASCADE
cascadeData	boolean	flag for DATA or NODATA
cascadeKey	boolean	flag for KEY or NOKEY
cascadePath	boolean	flag for PATH or NOPATH
logFlag	boolean	flag for LOG or NOLOG
segmentName	character	name of segment (NAME= keyword).
maximumLength	integer	length of segment (BYTES= keyword).
frequency	character	segment frequency (FREQ= keyword).
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
segmCaptureRoutine	character	name of data capture exit (for EXIT= keyword)
segmExitData	boolean	flag for DATA or NODATA
segmExitPath	boolean	flag for PATH or NOPATH
segmExitKey	boolean	flag for KEY or NOKEY
segmCascade	boolean	flag for CASCADE or NOCASCADE
segmCascadeData	boolean	flag for DATA or NODATA
segmCascadeKey	boolean	flag for KEY or NOKEY
segmCascadePath	boolean	flag for PATH or NOPATH
segmLogFlag	boolean	flag for LOG or NOLOG
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15
DAQdedbDBD		
dbdName	character	name of Data Base Definition (DBD)
versionString	character	value for VERSION= keyword
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
accessMethod	character	DBD access method - expected to be DEDB for this view
rmName	character	name of randomizing module
stage	integer	randomizing stage
extendedCall	boolean	randomizing XCI
ddName1	character	DDname for input file (DD1= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
areaSize	integer	Input record size (first parameter for SIZE= keyword)
root	integer	Root bytes (first parameter for ROOT= keyword)
rootOverflow	integer	Root overflow (second parameter for ROOT= keyword)
uow	integer	Units of work (first parameter for UOW= keyword)
uowOverflow	integer	Units of work (second parameter for UOW= keyword)
captureRoutine	character	name of data capture exit (for EXIT= keyword)
exitData	boolean	flag for DATA or NODATA
exitPath	boolean	flag for PATH or NOPATH
exitKey	boolean	flag for KEY or NOKEY
cascade	boolean	flag for CASCADE or NOCASCADE
cascadeData	boolean	flag for DATA or NODATA
cascadeKey	boolean	flag for KEY or NOKEY
cascadePath	boolean	flag for PATH or NOPATH
logFlag	boolean	flag for LOG or NOLOG
segmentName	character	name of segment (NAME= keyword).
parentSegmName	character	name of parent segment (PARENT= keyword).
exitFlag	boolean	flag for EXIT=NO

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
maximumLength	integer	length of segment (BYTES= keyword).
minimumLength	integer	minimum length of segment (BYTES= keyword).
rules	character	placement rules (RULES= keyword).
directDependent	boolean	segment type (TYPE= keyword). TRUE for DIR, FALSE for SEQ, ignored on root segment.
subsetPointers	integer	number of subset pointers (SSPTRS= keyword). Ignored unless directDependent=TRUE.
routine	character	name of compression routine (for COMPRTN= keyword)
initialization	boolean	flag for INIT subparameter on COMPRTN= keyword
dataOnly	boolean	flag for DATA or KEY subparameter on COMPRTN= keyword. TRUE is DATA, FALSE is KEY.
segmCaptureRoutine	character	name of data capture exit (for EXIT= keyword)
segmExitData	boolean	flag for DATA or NODATA
segmExitPath	boolean	flag for PATH or NOPATH
segmExitKey	boolean	flag for KEY or NOKEY
segmCascade	boolean	flag for CASCADE or NOCASCADE
segmCascadeData	boolean	flag for DATA or NODATA
segmCascadeKey	boolean	flag for KEY or NOKEY
segmCascadePath	boolean	flag for PATH or NOPATH
segmLogFlag	boolean	flag for LOG or NOLOG
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-
DAQhdamDBD		
dbdName	character	name of Data Base Definition (DBD)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
accessMethod	character	DBD access method - expected to be HDAM for this view
versionString	character	value for VERSION= keyword
osAccess	boolean	Operating system access method - TRUE for VSAM, false for OSAM
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO. Value is ignored when osAccess is FALSE.
rmName	character	name of randomizing module
relativeBlockNumber	integer	RBN value
rootAnchorPoints	integer	number of RAPS
rootMaxBytes	integer	Bytes value
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
datasetLabel	character	label for segment reference
ddName1	character	DDname for input file (DD1= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
size1	integer	Input record size (first parameter for SIZE= keyword)
freeBlockFrequency	integer	fbff value
freeSpacePercentage	integer	fspf value
primaryBlockingFactor	integer	Input blocking factor(BLOCK= keyword)
scanCylinders	integer	number of cylinders to search (SCAN= keyword)
searchAlgorithm	integer	search algorithm to use (SEARCHA= keyword)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
captureRoutine	character	name of data capture exit (for EXIT= keyword)
exitData	boolean	flag for DATA or NODATA
exitPath	boolean	flag for PATH or NOPATH
exitKey	boolean	flag for KEY or NOKEY
cascade	boolean	flag for CASCADE or NOCASCADE
cascadeData	boolean	flag for DATA or NODATA
cascadeKey	boolean	flag for KEY or NOKEY
cascadePath	boolean	flag for PATH or NOPATH
logFlag	boolean	flag for LOG or NOLOG
segmentName	character	name of segment (NAME= keyword).
parentSegmName	character	name of parent segment (PARENT= keyword).
pcPointer	character	SNGL or DBLE flag for physical parent pointer (second part of PARENT= keyword).
segmPointer	character	segment pointer - can be TWIN, TWINBWD, HIER, HIERBWD, NOTWIN (first part of POINTER= keyword).
ddName	character	DDname of dataset in which this segment is stored.
maximumLength	integer	length of segment (BYTES= keyword).
minimumLength	integer	length of segment (second subparameter for BYTES= keyword).
rules	character	placement rules - can be FIRST, LAST, HERE (RULES= keyword).
deleteRule	character	delete rule can be V, L, P, B (RULES= keyword).
replaceRule	character	replace rule can be V, L, P (RULES= keyword).
insertRule	character	insert rule can be V, L, P (RULES= keyword).
routine	character	name of compression routine (for COMPRTN= keyword)
initialization	boolean	flag for INIT subparameter on COMPRTN= keyword
dataOnly	boolean	flag for DATA or KEY subparameter on COMPRTN= keyword. TRUE is DATA, FALSE is KEY.
segmCaptureRoutine	character	name of data capture exit (for EXIT= keyword)
segmExitData	boolean	flag for DATA or NODATA
segmExitPath	boolean	flag for PATH or NOPATH
segmExitKey	boolean	flag for KEY or NOKEY

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
segmCascade	boolean	flag for CASCADE or NOCASCADE
segmCascadeData	boolean	flag for DATA or NODATA
segmCascadeKey	boolean	flag for KEY or NOKEY
segmCascadePath	boolean	flag for PATH or NOPATH
segmLogFlag	boolean	flag for LOG or NOLOG
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
indexDBDname	character	name of secondary index DBD
indexSegmName	character	name of secondary index segment
ddataFieldName	character	name of duplicate data fields on XDFLD statement.
searchFieldName	character	name of search fields on XDFLD statement.
subseqFieldName	character	name of subsequence fields on XDFLD statement.
sourceSegmName	character	name of index source segment on XDFLD statement.
exitRoutine	character	name of suppression routine on XDFLD statement.
constant	character	value for CONST keyword on XDFLD statement.
nullValue	character	value for NULLVAL keyword on XDFLD statement.
symbolic	boolean	value for POINTER keyword on LCHILD statement. TRUE is SYMB, FALSE is INDX.
xdfldName	character	name of XDFLD statement.
lchildDBDname	character	name of logical child DBD.
lchildSegmName	character	name of logical segment DBD.
lchildPairDBDname	character	name of DBD for logical pair.
lchildPairSegmName	character	name of logical pair segment (PAIR= on LCHILD statement).
lcPointer	character	value for POINTER keyword on LCHILD statement - values are SNGL, DBLE, NONE
rules	character	value for RULES keyword - values are FIRST, LAST, HERE

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
lparentDBDname	character	name of logical parent DBD.
lparentSegmName	character	name of logical parent DBD.
counter	boolean	flag to specify addition of CTR to segment POINTER= keyword
lparent	boolean	flag to specify addition of LPARNT to segment POINTER= keyword
ltwin	character	value of logical twin pointers for segment POINTER= keyword - can be LTWIN or LTWINBWD
virtualParent	boolean	VIRTUAL or PHYSICAL for logical parent statement
pairDBDname	character	name of DBD that has segment paired to this segment
pairSegmName	character	name of segment paired to this segment (SOURCE= keyword).
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-
DAQhidamDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be HDAM for this view
versionString	character	value for VERSION= keyword
osAccess	boolean	Operating system access method - TRUE for VSAM, false for OSAM
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO. Value is ignored when osAccess is FALSE.
primaryDBDname	character	name of primary index DBD
primarySegmName	character	name of primary index segment
rootFieldName	character	name of root segment sequence field
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
datasetLabel	character	label for segment reference

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
ddName1	character	DDname for input file (DD1= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
size1	integer	Input record size (first parameter for SIZE= keyword)
freeBlockFrequency	integer	fbff value
freeSpacePercentage	integer	fspf value
primaryBlockingFactor	integer	Input blocking factor(BLOCK= keyword)
scanCylinders	integer	number of cylinders to search (SCAN= keyword)
searchAlgorithm	integer	search algorithm to use (SEARCHA= keyword)
captureRoutine	character	name of data capture exit (for EXIT= keyword)
exitData	boolean	flag for DATA or NODATA
exitPath	boolean	flag for PATH or NOPATH
exitKey	boolean	flag for KEY or NOKEY
cascade	boolean	flag for CASCADE or NOCASCADE
cascadeData	boolean	flag for DATA or NODATA
cascadeKey	boolean	flag for KEY or NOKEY
cascadePath	boolean	flag for PATH or NOPATH
logFlag	boolean	flag for LOG or NOLOG
segmentName	character	name of segment (NAME= keyword).
parentSegmName	character	name of parent segment (PARENT= keyword).
pcPointer	character	SNGL or DBLE flag for physical parent pointer (second part of PARENT= keyword).
segmPointer	character	segment pointer - can be TWIN, TWINBWD, HIER, HIERBWD, NOTWIN (first part of POINTER= keyword).
ddName	character	DDname of dataset in which this segment is stored.
maximumLength	integer	length of segment (BYTES= keyword).
minimumLength	integer	length of segment (second subparameter for BYTES= keyword).

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
rules	character	placement rules - can be FIRST, LAST, HERE (RULES= keyword).
deleteRule	character	delete rule can be V, L, P, B (RULES= keyword).
replaceRule	character	replace rule can be V, L, P (RULES= keyword).
insertRule	character	insert rule can be V, L, P (RULES= keyword).
routine	character	name of compression routine (for COMPRTN= keyword)
initialization	boolean	flag for INIT subparameter on COMPRTN= keyword
dataOnly	boolean	flag for DATA or KEY subparameter on COMPRTN= keyword. TRUE is DATA, FALSE is KEY.
segmCaptureRoutine	character	name of data capture exit (for EXIT= keyword)
segmExitData	boolean	flag for DATA or NODATA
segmExitPath	boolean	flag for PATH or NOPATH
segmExitKey	boolean	flag for KEY or NOKEY
segmCascade	boolean	flag for CASCADE or NOCASCADE
segmCascadeData	boolean	flag for DATA or NODATA
segmCascadeKey	boolean	flag for KEY or NOKEY
segmCascadePath	boolean	flag for PATH or NOPATH
segmLogFlag	boolean	flag for LOG or NOLOG
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
indexDBDname	character	name of secondary index DBD
indexSegmName	character	name of secondary index segment
ddataFieldName	character	name of duplicate data fields on XDFLD statement.
searchFieldName	character	name of search fields on XDFLD statement.
subseqFieldName	character	name of subsequence fields on XDFLD statement.
sourceSegmName	character	name of index source segment on XDFLD statement.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
exitRoutine	character	name of suppression routine on XDFLD statement.
constant	character	value for CONST keyword on XDFLD statement.
nullValue	character	value for NULLVAL keyword on XDFLD statement.
symbolic	boolean	value for POINTER keyword on LCHILD statement. TRUE is SYMB, FALSE is INDX.
xdfldName	character	name of XDFLD statement.
lchildDBDname	character	name of logical child DBD.
lchildSegmName	character	name of logical segment DBD.
lchildPairDBDname	character	name of DBD for logical pair.
lchildPairSegmName	character	name of logical pair segment (PAIR= on LCHILD statement).
lcPointer	character	value for POINTER keyword on LCHILD statement - values are SNGL, DBLE, NONE
rules	character	value for RULES keyword - values are FIRST, LAST, HERE
lparentDBDname	character	name of logical parent DBD.
lparentSegmName	character	name of logical parent DBD.
counter	boolean	flag to specify addition of CTR to segment POINTER= keyword
lparent	boolean	flag to specify addition of LPARNT to segment POINTER= keyword
ltwin	character	value of logical twin pointers for segment POINTER= keyword - can be LTWIN or LTWINBWD
virtualParent	boolean	VIRTUAL or PHYSICAL for logical parent statement
pairDBDname	character	name of DBD that has segment paired to this segment
pairSegmName	character	name of segment paired to this segment (SOURCE= keyword).
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-
DAQhisamDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be HDAM for this view
versionString	character	value for VERSION= keyword

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO.
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
ddName1	character	DDname for primary file (DD1= keyword)
ddName2	character	DDname for overflow file (OVFLW= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
inputRecordLength	integer	Record length of primary file (first parameter for RECLEN= keyword)
outputRecordLength	integer	Record length of overflow file (second parameter for RECLEN= keyword)
size1	integer	Primary record size (first parameter for SIZE= keyword)
size2	integer	Overflow record size (second parameter for SIZE= keyword)
primaryBlockingFactor	integer	Primary file blocking factor (first parameter for BLOCK= keyword)
overflowBlockingFactor	integer	Overflow file blocking factor (second parameter for BLOCK= keyword)
captureRoutine	character	name of data capture exit (for EXIT= keyword)
exitData	boolean	flag for DATA or NODATA
exitPath	boolean	flag for PATH or NOPATH
exitKey	boolean	flag for KEY or NOKEY
cascade	boolean	flag for CASCADE or NOCASCADE
cascadeData	boolean	flag for DATA or NODATA
cascadeKey	boolean	flag for KEY or NOKEY

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
cascadePath	boolean	flag for PATH or NOPATH
logFlag	boolean	flag for LOG or NOLOG
segmentName	character	name of segment (NAME= keyword).
parentSegmName	character	name of parent segment (PARENT= keyword).
pcPointer	character	SNGL or DBLE flag for physical parent pointer (second part of PARENT= keyword).
segmPointer	character	segment pointer - can be TWIN, TWINBWD, HIER, HIERBWD, NOTWIN (first part of POINTER= keyword).
ddName	character	DDname of dataset in which this segment is stored.
maximumLength	integer	length of segment (BYTES= keyword).
minimumLength	integer	length of segment (second subparameter for BYTES= keyword).
frequency	character	segment frequency (FREQ= keyword).
rules	character	placement rules - can be FIRST, LAST, HERE (RULES= keyword).
deleteRule	character	delete rule can be V, L, P, B (RULES= keyword).
replaceRule	character	replace rule can be V, L, P (RULES= keyword).
insertRule	character	insert rule can be V, L, P (RULES= keyword).
routine	character	name of compression routine (for COMPRTN= keyword)
initialization	boolean	flag for INIT subparameter on COMPRTN= keyword
dataOnly	boolean	flag for DATA or KEY subparameter on COMPRTN= keyword. TRUE is DATA, FALSE is KEY.
segmCaptureRoutine	character	name of data capture exit (for EXIT= keyword)
segmExitData	boolean	flag for DATA or NODATA
segmExitPath	boolean	flag for PATH or NOPATH
segmExitKey	boolean	flag for KEY or NOKEY
segmCascade	boolean	flag for CASCADE or NOCASCADE
segmCascadeData	boolean	flag for DATA or NODATA
segmCascadeKey	boolean	flag for KEY or NOKEY
segmCascadePath	boolean	flag for PATH or NOPATH
segmLogFlag	boolean	flag for LOG or NOLOG
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
indexDBDname	character	name of secondary index DBD
indexSegmName	character	name of secondary index segment
ddataFieldName	character	name of duplicate data fields on XDFLD statement.
searchFieldName	character	name of search fields on XDFLD statement.
subseqFieldName	character	name of subsequence fields on XDFLD statement.
sourceSegmName	character	name of index source segment on XDFLD statement.
exitRoutine	character	name of suppression routine on XDFLD statement.
constant	character	value for CONST keyword on XDFLD statement.
nullValue	character	value for NULLVAL keyword on XDFLD statement.
symbolic	boolean	value for POINTER keyword on LCHILD statement. TRUE is SYMB, FALSE is INDX.
xdfldName	character	name of XDFLD statement.
lchildDBDname	character	name of logical child DBD.
lchildSegmName	character	name of logical segment DBD.
lchildPairDBDname	character	name of DBD for logical pair.
lchildPairSegmName	character	name of logical pair segment (PAIR= on LCHILD statement).
lcPointer	character	value for POINTER keyword on LCHILD statement - values are SNGL, DBLE, NONE
rules	character	value for RULES keyword - values are FIRST, LAST, HERE
lparentDBDname	character	name of logical parent DBD.
lparentSegmName	character	name of logical parent DBD.
counter	boolean	flag to specify addition of CTR to segment POINTER= keyword
lparent	boolean	flag to specify addition of LPARNT to segment POINTER= keyword

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
ltwin	character	value of logical twin pointers for segment POINTER= keyword - can be LTWIN or LTWINBWD
virtualParent	boolean	VIRTUAL or PHYSICAL for logical parent statement
pairDBDname	character	name of DBD that has segment paired to this segment
pairSegmName	character	name of segment paired to this segment (SOURCE= keyword).
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-
DAQindexDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be HDAM for this view
versionString	character	value for VERSION= keyword
passwordFlag	boolean	Flag for PASSWD keyword. TRUE for PASSWD=YES, FALSE for PASSWD=NO.
dosCompatibility	boolean	Flag for DOSCOMP keyword.
protect	boolean	Flag for PROTECT keyword.
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
usesDataStructure	character	name of ShareableDataStructure that maps a segment in this DBD.
usesDataElement	character	name of ShareableDataElement that is used as a Field by this DBD.
ddName1	character	DDname for primary file (DD1= keyword)
ddName2	character	DDname for overflow file (OVFLW= keyword)
dsDevice	character	Type of data storage hardware (DEVICE= keyword). Supported values are 2305, 2319, 3330, 3340, 3350, 3375, 3380,
dsModel	character	Model of data storage hardware (MODEL= keyword). Supported values are ″1″, ″2″, ″11″. The value here is relevant only when the dsDevice is ″3330″ or ″2305″.
inputRecordLength	integer	Record length of primary file (first parameter for RECLEN= keyword)

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
outputRecordLength	integer	Record length of overflow file (second parameter for RECLEN= keyword)
size1	integer	Primary record size (first parameter for SIZE= keyword)
size2	integer	Overflow record size (second parameter for SIZE= keyword)
primaryBlockingFactor	integer	Primary file blocking factor (first parameter for BLOCK= keyword)
overflowBlockingFactor	integer	Overflow file blocking factor (second parameter for BLOCK= keyword)
segmentName	character	name of segment (NAME= keyword).
maximumLength	integer	length of segment (BYTES= keyword).
frequency	character	segment frequency (FREQ= keyword).
mappingStruct..DAQDataStructure		relationship to shareable data structure that maps the segment.
segmLabel	character	type of description
segmDescription	character	description of segment
logSegmName	character	name of logical segment that uses this segment as its source.
logDBDname	character	name of logical DBD with a segment that uses this segment as its source.
hidamDBDname	character	name of primary index target DBD
hidamRootName	character	name of primary index target segment
indexedFieldName	character	name of sequence field in primary index target segment
targetDBDname	character	name of primary index target DBD
targetSegmName	character	name of primary index target segment
ddataFieldName	character	name of duplicate data fields on XDFLD statement.
searchFieldName	character	name of search fields on XDFLD statement.
subseqFieldName	character	name of subsequence fields on XDFLD statement.
sourceSegmName	character	name of index source segment on XDFLD statement.
exitRoutine	character	name of suppression routine on XDFLD statement.
constant	character	value for CONST keyword on XDFLD statement.
nullValue	character	value for NULLVAL keyword on XDFLD statement.

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
symbolic	boolean	value for POINTER keyword on LCHILD statement. TRUE is SYMB, FALSE is INDX.
xdfldName	character	name of XDFLD statement.
fieldName	character	name of field (NAME= keyword).
sequenceField	boolean	flag defining whether the field is a sequence field (SEQ keyword)
uniqueSequence	boolean	Flag defining whether the sequence field is unique. TRUE corresponds to parameter value of 'U'. This value is ignored when sequenceField is FALSE.
fieldLabel	character	type of description
fieldDescription	character	description of field
defaultName	character	default name of data item attached to this field.
offsetFromLevel01	integer	1 less than the start position for the field (STARTPOS= keyword).
pSharedElement..DAQDataElement		relationship to shareable data element that holds data for the data item.
pSharedStructure..DAQDataStructure		relationship to shareable data structure that holds data for the data item.
diName1	character	Local name for data item stored in a data element type of alias.
diName2	character	Local name for data item stored in a data element type of alias.
isConstant	boolean	flag
asBitData	boolean	flag indicating character string should be treated as bit (binary) data
genericPicture	character	language-independent picture
isComplex	boolean	flag indicating real or complex number
isSigned	boolean	flag indicating unsigned or signed number
isLOB	boolean	flag indicating string is a large object (LOB)
numericPrecision	smallint	numeric precision in bits or digits depending on data type
scale	smallint	position of binary or decimal point
strLength	integer	maximum length of string data in bits, bytes or double-bytes
strVaryingFlag	integer	indication of variable length string: 0-varying, 1-varyingZ (PL/I), 2-nonvarying

Table 7. Views and Attributes of Object Types (continued)

View / Attribute Name	Data Type	Description
typeCode	integer	data type code: 0-binary number, 1-packed decimal, 2-zoned decimal, 3-binary floating point, 4-decimal floating p string, 7-double-byte character, 8-mixed double/single-byte, 9-date, 10-time, 11-timestamp, 12-index (COBOL), 13-undefined, 14-rowid (Oracle), 15-
DAQlogicalDBD		
dbdName	character	name of Data Base Definition (DBD)
accessMethod	character	DBD access method - expected to be LOGICAL for this view
versionString	character	value for VERSION= keyword
pcbName	character	name of PCB based on this DBD
psbName	character	name of PSB containing a PCB based on this DBD
dbdLabel	character	type of description
dbdDescription	character	description of DBD
segmLabel	character	type of description
segmDescription	character	description of segment
segmentName	character	name of segment (NAME= keyword).
parentSegmName	character	name of parent segment (PARENT= keyword).
sourceSegmName	character	name of physical segment that is the source of this segment.
sourceDBDname	character	name of physical DBD on which this DBD is based.

Appendix G. Actions and Rules

This appendix contains two tables that show the actions and underlying rules DataAtlas Designer associates with any object type and function (for example, “table” and “inform”) you submit to it.

The first table, “Object Types, Functions, and Actions,” associates only actions with object types and functions. The rightmost column of the table contains rule IDs that index into the second table, “Rule Explanations.” This table explains the rules that determine what DataAtlas Designer informs you of, what it considers invalid, what it proposes to do, and how it carries out proposals.

The “Rule Explanations” table also contains a “Version” column. It tells you the lowest level release of DB2/390 a rule applies to. In some cases, the column specifies that the rule applies to only one version of DB2/390.

Object Types, Functions, and Actions

Table 8. Object Types, Functions, and Actions

Object Type	Function	Action	Rule ID
Column	Inform	Identify columns with data type problems	COLI01 COLI02 COLI03
		Identify columns with missing design information	COLI04
Database	Inform	Identify databases with defaults	DBSI01 DBSI02
		Identify databases with performance problems	DBSI03 DBSI04

Table 8. Object Types, Functions, and Actions (continued)

Object Type	Function	Action	Rule ID
Index	Inform	Identify indexes that can result in increased maintenance cost	INXI01 INXI02 INXI03 INXI04
		Identify indexes that could be dropped	INXI05
		Identify Indexes with default values	INXI06
	Propose	Calculate space requirements	INXP01 INXP02 INXP03 INXP04 INXP05 INXP06
		Set options	INXP07
	Validate	Identify indexes with column inconsistencies	INXV01 INXV02 INXV03
Storage group	Inform	Identify storage groups with SMS usage	STGI01
	Propose	Calculate required space	STGP01
	Validate	Identify storage groups with invalid definitions	STGV01 STGV02
		Identify incomplete storage groups	STGV03 STGV04

Table 8. Object Types, Functions, and Actions (continued)

Object Type	Function	Action	Rule ID
Table	Inform	Identify tables with default values	TBLI01 TBLI02
		Identify tables with potential column problems	TBLI03
		Identify tables with potential primary key problems	TBLI04 TBLI05 TBLI06
		Identify tables with missing design problems	TBLI07 TBLI08
		Identify tables with an explicit clustering index	TBLI09
		Identify large tables	TBLI10
		Calculate wasted space	TBLI11
	Propose	Assign table to table space	TBLP01 TBLP02 TBLP03 TBLP04
		Create primary key	TBLP05
		Create foreign keys	TBLP06
		Create index	TBLP07 TBLP08 TBLP09 TBLP10 TBLP11 TBLP12 TBLP13 TBLP14 TBLP15
		Create partitioning index for partitioned table	TBLP16
	Validate	Identify tables with invalid primary/foreign keys	TBLV01 TBLV02 TBLV03 TBLV04 TBLV05 TBLV06 TBLV07 TBLV08 TBLV09
		Identify tables with column inconsistencies	TBLV10 TBLV11
		Identify incomplete tables	TBLV12

Table 8. Object Types, Functions, and Actions (continued)

Object Type	Function	Action	Rule ID
Table spaces	Inform	Identify table spaces with default values	TSPI01 TSPI02 TSPI03 TSPI04 TSPI05 TSPI06 TSPI07 TSPI08
		Identify table spaces with design inconsistencies	TSPI09 TSPI10 TSPI11 TSPI12 TSPI13 TSPI14
	Propose	Calculate space requirements	TSPP01 TSPP02 TSPP03 TSPP04
		Calculate the segment size	TSPP05
		Set options	TSPP06 TSPP07 TSPP08 TSPP09 TSPP10
	Validate	Identify Invalid table spaces assigned to DS NDB07	TSPV01
		Identify invalid table spaces	TSPV02 TSPV03
		Identify incomplete table spaces	TSPV05 TSPV05

Rule Explanations

Table 9. Rule Explanations

Rule ID	Version	Rule explanation
COLI01	V3	Tables are identified that have variable-length columns with a length of <18> bytes. When you use a variable-length column, the range should be sufficiently large. If the variable-length is less than or equal to <18> it is recommend that you use a fixed-length column instead of a variable-length column. (Valid: 1 - 127).

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
COLI02	V3	Tables are identified that have variable-length columns for which the difference between maximum and average length is too low. It is recommended that you use a fixed-length column instead of a variable-length column if the difference between the average length and the maximum length of variable-length column is 20% of its maximum length, and the maximum length is less than or equal to 254 bytes.
COLI03	V3	Tables are identified that have columns with data type GRAPHIC, VARGRAPHIC, or LONGVARGRAPIC. These data types should only be used if a column contains double-byte character.
COLI04	V3	Missing data load or work load is identified for the columns of a table. Some design proposals cannot be processed if data load or work load specifications are incomplete.
DBSI01	V3	If no buffer pool is specified, the DB2 default (BP0) is used.
DBSI02	V3	If no storage group is specified, the DB2 default (SYSDEFLT) is used.
DBSI03	V3	If a database has table spaces assigned with usage intent QMF and the database usage intent is different, there can be performance problems. QMF users who create their own tables should be allowed to maintain their own databases.
DBSI04	V3	If the buffer pool assigned to the database has no size value, or if the buffer pool is a 32KB buffer pool, there can be performance problems. If the buffer pool is a 32KB buffer pool, BP0 is used as default for all assigned indexes. For all other tables, a 32KB buffer has a negative performance impact. If the assigned buffer pool has no size value, you should check if table spaces or indexes using this buffer pool should be reassigned to an appropriate buffer pool.
INXI01	V3	Index columns with an UPDATE frequency greater than X (X=<6>) are identified. If index columns with an UPDATE frequency greater than X (X=<6>) (Valid: 1 - 10) are used, maintenance could be more time and cost consuming.
INXI02	V3	Index columns with variable-length data types are identified. Maintenance is more time and cost consuming if the index columns have these variable-length data types: VARCHAR, LONG VARCHAR, VARGRAPHIC, LONG VARGRAPHIC.
INXI03	V3	Index are identified whose row length is greater than 40 bytes. Maintenance is more time and cost consuming if the index row length is greater than 40 bytes (Valid: 1 - 111). Index columns longer than 40 bytes build multilevel indexes and require additional I/O.
INXI04	V3	Indexes are identified where the number of referenced table rows is very small and where there is only little access to them. An index should be dropped if the table is too small and if the table frequency values are too low for the UPDATE, INSERT, DELETE and SELECT operation.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
INXI05	V3	Index columns are identified whose number of distinct values is too low. An index should be dropped if the number of distinct values for the first column is less that <10> (valid: 1 - 999999999) for a nonclustered index.
INXI06	V4	The index type has not been specified. The installation default (if any) or type 2 will be assumed.
INXP01	V3	The FREEPAGE values for the index space is calculated based on the average workload of the corresponding table. If table design information is missing, the default is proposed, Default: <0> (Valid: 0 - 255)
INXP02	V3	The PCTFREE value for the index space is calculated based on the workload of the corresponding table. If table design information is missing, the default is proposed. Default: <10> (Valid: 0 - 99)
INXP03	V3	The number of subpages is calculated based on the workload of the corresponding table and the usage intent of the corresponding table space. If the table work load information is incomplete or if the table space usage intent is missing, the default is used for the SUBPAGES value. Default for a type 1 index: <4> (Valid: 1, 2, 4, 8, 16). For a type 2 index, the number of subpages is always 1.
INXP04	V3 Only	The PRIQTY value for the index space is calculated based on the index length, the number of SUBPAGES, the values for FREEPAGE and PCTFREE, the dataload of the corresponding table, and the number of distinct values of the index columns. If you did not also request the calculation of the FREEPAGE, PCTFREE, or SUBPAGES values with this proposal, the proposed PRIQTY value is based on the current table space values for FREEPAGE, PCTFREE, and SUBPAGES. Default: <12> KB (Valid: 12 - 4194304)
INXP05	V3	The SECQTY value for the index space is calculated based on the PRIQTY value and the confidence factor. If you did not request the proposal for the SECQTY value, the proposed SECQTY value is based on current table space value for PRIQTY. Default: <12> KB (Valid: 12 - 131068)
INXP06	V4	The PRIQTY value for the index space is calculated based on the index length, the values for FREEPAGE and PCTFREE, the dataload of the corresponding table, and the number of distinct values of the index columns. If you did not also request the calculation of the FREEPAGE or PCTFREE with this proposal, the proposed PRIQTY value is based on the current table space values for FREEPAGE and PCTFREE. The PRIQTY value has been adjusted for large table spaces in DB2/390 Version 5. Default: <12> KB (Valid: 12 - 4194304)
INXP07	V3	The ERASE option of the index is set if the Security option is set for the corresponding table. The ERASE option ensures that, before the index is dropped, DB2 overwrites with zeros all data sets related to the index that may contain confidential data.
INXV01	V3	An index is invalid if the number of columns exceeds 64.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
INXV02	V3 Only	An index is invalid if the index row length exceeds the maximum row length. The maximum row length depends on the number of SUBPAGES and the UNIQUE indicator.
INXV03	V4	An index is invalid if the index row length exceeds the maximum row length.
STGI01	V3	If the VOLUMES attribute is set to '*', the Storage Management System (SMS) is used. In this case, SMS manages all data sets created for the storage group.
STGP01	V3	The required space value for a storage group is calculated, based on the storage requirement values of the table spaces and indexes assigned to the storage group. The storage requirement is defined as : PRIQTY + (<1> * SECQTY). (Valid: 0 - 999999999) If PRIQTY and SECQTY values are not available, defaults are used.
STGV01	V3	A storage group definition is invalid if a volume serial number is specified more than once.
STGV02	V3	A storage group definition is invalid if the attribute VOLUMES is set '*' and the usage of the Storage Management System (SMS) is set to 'NO'. An asterisk for the VOLUMES attribute indicates the usage of the Storage Management System (SMS). On the other hand, the specification of the volume serial numbers indicates storage management by the user.
STGV03	V3	A storage group is incomplete if the required space value is not yet calculated. The value is used as a base for requesting volume IDs.
STGV04	V3	A storage group is incomplete if no volumes are defined for the storage group and SMS is not used.
TBLI01	V3	Tables are identified for which the AUDIT option is not specified. If the AUDIT option is not specified, the DB2 default NONE is used. No audit trail is produced.
TBLI02	V3	Tables are identified for which the DATA CAPTURE option is not specified. If the DATA CAPTURE option is not specified, the DB2 default NONE is used.
TBLI03	V3	Tables are identified in which variable-length columns precede fixed-length columns. When variable-length columns occur at the beginning of a row, it is more difficult to calculate the positions of subsequent fixed-length columns. It is recommended that you change the column order such that the variable-length columns occur at the end of a row.
TBLI04	V3	Tables without a primary key are identified. To establish relationships between parent tables and dependent tables, you need to specify a primary key in parent tables.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TBLI05	V3	Primary keys with default values are identified. Primary key columns defined as 'NOT NULL WITH DEFAULT' and with a data type other that TIMESTAMP can cause problems, because only one zero or blank row with default values is allowed.
TBLI06	V3	Primary key columns with an UPDATE frequency greater than 0 are identified. Avoid using primary key columns that are updated because they can cause integrity constraint problems at run time. Primary key values in rows that have at least one dependent must not be changed.
TBLI07	V3	Tables or table partitions with missing data load or work load information are identified. Some design proposals cannot be processed if data load or work load specifications are incomplete.
TBLI08	V3	Tables are identified for which the Security option is not set. It is possible to process the proposal for the ERASE option of the table's table space and index only if the Security option is set for a table.
TBLI09	V3	Tables are identified for which no explicit clustered index is created. If none of the table's indexes is explicitly created as clustered, DB2 defines the first index of a table as clustered.
TBLI10	V3	The number of table space pages required by a table is calculated. If the required space of a table exceeds <10000> pages, it is recommended that you use a partitioned table space for it.
TBLI11	V3	The total amount of wasted space is calculated using the following input values: • Usable page size (4K or 32K pages) • Initial number of rows per table (required) • Growth rate, DELETE rate, and maintenance period (optional) • PCTFREE, FREEPAGE (optional) • Compress option set or not set • Row length of the table (calculated) If any of the required information is missing, wasted space cannot be calculated.
TBLP01	V3	If n partitions are defined for the table, a table space with n partitions is created. The partitioned table is assigned to the partitioned table space.
TBLP02	V3	A medium-sized table is assigned to a single-table segmented table space. A medium sized table is assumed to be a table with a storage requirement of between <1000> and <10000> pages. (Valid: 1 - 999999999). The required pages are calculated based on the value for the initial number of rows and the table's row length.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TBLP03	V3	If a small table that is not partitioned has key relationships to tables that are assigned to multi-table segmented table spaces, these table space are proposed for assignment of the small table. From the list of proposed table spaces, select the table space to which you want to assign the small table. For a small table without any key relationships, a new table space is proposed. The size of a small table is less than the lower limit of a medium-sized table.
TBLP04	V3	By default, a table is assigned to a simple table space.
TBLP05	V3	For the creation of a primary key, all of the unique columns are proposed. From the list of proposed unique columns, select the columns on which you want to define the primary key.
TBLP06	V3	For the creation of a foreign key, a list of potential foreign keys is proposed. Potential foreign keys fulfill the following criteria: • If they have the same data type as primary key columns of another table • If they have the same number of distinct values as primary key columns of another table. From the list of proposed foreign keys, select the appropriate ones.
TBLP07	V3	Filter criterion: If you selected to create an index, create the index only on columns with the highest number of distinct values. If no design information is available, all columns have the same priority. This rule does not apply to the creation of partitioning indexes, or when an index is created on the basis of existing primary, foreign, or unique indexes.
TBLP08	V3	Filter criterion: If you selected to create an index, create the index only if the number of table rows is greater than X (X=<100>) and the number of existing indexes is less than Y (Y=<5>) (Valid: 1 999999999). The values of X and Y depend on the relationships between the table frequency values for the INSERT, DELETE, and SELECT operation.
TBLP09	V3	If a table has a primary key, a unique index is created on all primary key columns.
TBLP10	V3	For each unique key of a table, a unique index is created on all of the corresponding unique key columns.
TBLP11	V3	Filter criterion: If you selected to create and index, create the index only on the columns with the lowest UPDATE frequency. Columns without design information on the update rate have the lowest priority. This rule does not apply to the creation of partitioning indexes. This rule does not apply to the creation of partitioning indexes or when an index is created on the basis of existing primary foreign or unique keys.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TBLP12	V3	If no clustered index is created on the table, a clustered index is created on the columns of the foreign key with the maximum frequency of join operations on its columns. A nonclustered index is created on the columns of the remaining foreign keys.
TBLP13	V3	An index is created on columns with a high value for the frequency of the Sorting operation. (Value > <6>). Valid 0 - 10).
TBLP14	V3	An index is created on columns frequently used in join predicates. (Value > <3>). (Valid 0 - 10).
TBLP15	V3	An index is created on columns with a high value for the frequency of the WHERE clause. (Value > <3>). (Valid 0 - 10).
TBLP16	V3	If a partitioned table is assigned to a partitioned table space, a partitioning clustered index is created. A partitioning clustered index can only be created if the table and table space have the same number of partitions.
TBLV01	V3	The primary key of a table is invalid if there is no unique index on the primary key columns.
TBLV02	V3	The key of a table (primary, foreign, or unique) is invalid if the number of key columns exceeds 64.
TBLV03	V3	The primary key or the unique key of a table is invalid if the sum of the length attributes exceeds 254.
TBLV04	V3	The foreign key of a table is invalid if the sum of the length attributes exceeds 254-n, where n is the number of columns that allow NULL values.
TBLV05	V3	The foreign key of a table is invalid if the referenced table has no primary key.
TBLV06	V3	The foreign key of a table is invalid if the number of foreign key columns do not match the number of primary key columns of the parent table.
TBLV07	V3	The foreign key of a table is invalid if the program name of the field procedures exit routine (FieldProc) and the parameters passed on to it differ between the foreign key columns and the primary key columns of the parent table.
TBLV08	V3	The foreign key of a table is invalid if the data types of the foreign key columns do not match the data types of the primary key columns of the parent table.
TBLV09	V3	The primary key of a table is invalid if one of the primary key columns allows NULL values.
TBLV10	V3	The definition of a table is invalid if the sum of byte count values of the columns exceeds the table maximum row size. The maximum row size of a table depends on the page size of the assigned buffer pool and on whether the edit procedure (EditProc) is specified.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TBLV11	V3	The definition of a column is invalid if the column data type is VARCHAR with a length value greater than 254 **and** the field procedure exit routine FieldProc is specified, or if the column data type is VARGRAPHIC with a length value greater than 127 **and** the field procedure exit routine ('FieldProc') is specified.
TBLV12	V3	The definition of a table is incomplete if no columns are defined for the table.
TSPI01	V3	If the table space is not assigned to a database, the DB2 default database DSNDB04 is used.
TSPI02	V3	If no buffer pool is specified for the table space, the buffer pool of the database is used. If the database has no buffer pool the DB2 default buffer pool BP0 is used.
TSPI03	V3	If the locking size value is not specified, the DB2 default ANY is used.
TSPI04	V3	If neither storage group not VCAT is specified, the storage group assigned to the database is used. If no storage group is assigned to the database, the DB2 default storage group SYSDEFLT is used. Note that if you use the default storage group, the table space is restricted to the default space allocation.
TSPI05	V3	If the Close option is not specified, the DB2 default NO (not set) is used. If the limit of open files in your DB2 subsystem is reached, you should set the Close option.
TSPI06	V3	If the Compress option is not specified, the DB2 default NO (not set) is used. Before you set the compress option, you should carefully consider the following: • Compression increases processing costs. • Only tables with similar kinds of data should be put into the same table space. • For small table spaces, the size of the compression dictionary (8 KB to 64 KB) can offset the space savings provided by the compression.
TSPI07	V3	If FREEPAGE is not specified, the DB2 default 0 is used. For read-only tables, FREEPAGE 0 is appropriate. For tables with a high INSERT frequency or updates that lengthen the row, a higher amount of FREEPAGE is useful because the table rows are stored closer together. This applies especially to tables that have columns with variable-length data types. The drawback is the higher amount of space needed.
TSPI08	V3	Table space with a default PCTFREE value are identified. If PCTFREE is not specified, the DB2 default 5 is used. For read-only tables, PCTFREE is appropriate. For tables with a higher INSERT frequency, a higher amount of PCTFREE is useful, because the table rows will be stored together. This applies especially to tables that have columns with variable-length data types. The drawback is the higher amount of space needed.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TSPI09	V3	A table space has a potential design problem if a simple table space has more than one table assigned. Consider using a segmented table space instead.
TSPI10	V3	The database DSNDB07 is a special DB2 database for temporary work file table spaces. They are used by DB2 for sorting. To distribute I/Os, each temporary work file table space should be assigned to a different volume. Other frequently used data sets should not be placed on the same volume. In addition to the defaults set by DB2, additional temporary work file table space should be defined.
TSPI11	V3	A table space assigned to storage group SYSDEFLT should only be used for prototyping or test purposes.
TSPI12	V3	Partitioned table spaces with no matching partitioned tables are identified. The partitioning information of a table space should be checked if a partitioned table space has a NUMPARTS value that does not match the number of table partitions of the assigned table.
TSPI13	V4	All indexes on all tables in a table space must be a type 2 to use row-level locking. A table space has a potential design problem if LOCKSIZE=ROW is specified and any index defined on any table in the table space is not a type 2 index.
TSPI14	V5	A type 1 index is not allowed for a table in a large table space.
TSPP01	V3	The FREEPAGE value is calculated based on the average workload of the assigned tables. Default: <0> (Valid: 0 - 255)
TSPP02	V3	The PCTFREE value is calculated based on the average workload of the assigned tables. Default: <5> (Valid: 0 - 99)
TSPP03	V3	THE PRIQTY value is calculated based on the space requirements of all assigned tables. The space requirement of a table depends on: the data load, the row length, FREEPAGE and PCTFREE values and the assigned buffer pool. If the calculation of the values for FREEPAGE and PCTFREE is not requested for this proposal, the proposed PRIQTY value is based on the current table space values for FREEPAGE and PCTFREE. Default: 4K pages, <12> KB; 32K pages, <96>KB (Valid: 12 - 4194304)
TSPP04	V3	THE SECQTY value is calculated based on the PRIQTY value and the confidence factor. If the calculation of the PRIQTY value is not requested with this proposal, the proposed SECQTY value is based on the current table space value for PRIQTY. Default: 4K pages, <12> KB; 32K pages, <96> KB (Valid: 12 - 131040)

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TSPP05	V3	A SEGSIZE value greater than 0 indicates that the table space is segmented. If the SEGSIZE value of an existing segmented table space is changed, the existing table space must be dropped and a new one must be created. The proposed SEGSIZE value depends on the size of the largest table in the table space. Size of table <= 28 pages : SEGSIZE 4 to 28 Size of table < 128 pages : SEGSIZE 32 Size of table >= 128 pages : SEGSIZE 64 Default <4> (Valid n*4 where n is 1 - 16)
TSPP06	V3	The Close option default is <NO>. (Valid: NO, YES)
TSPP07	V3	If the Security option is set for at least one table assigned to a simple or segmented table space, the ERASE option is set too. Otherwise, the ERASE option is NOT set. For partitioned table spaces the ERASE option is set according to the Security option setting in the table partition. If the Security option is not set for all tables assigned to the table space, the default value is set for the ERASE option. Default: <NO>
TSPP08	V3 Only	The Locking size value 'TABLESPACE' should be used if all assigned tables are read-only or if the concurrency is low. The Locking size value 'ANY' should be used if the concurrency is medium or high. Read only means that INSERT, UPDATE, and DELETE frequency values are 0. A partitioned table is read-only if all partitions are read-only. The concurrency of a partitioned table is low if the concurrency value on all partitions is low.
TSPP09	V4	Propose LOCKSIZE=TABLESPACE, if all assigned tables are read only, else, LOCKSIZE=ANY is proposed. Read only means, that insert, update and delete frequency values are 0; A partitioned table is read only if all partitions are read only. Propose LOCKSIZE=ROW, if all assigned tables are updated randomly. If design information is mising, the default value for the locking size is proposed which is ANY.
TSPP10	V3	If there is a table in the TS with row length > 4056 Bytes, a 32K buffer pool is proposed. If no buffer pools are available, the DB2 default buffer pool is used: Default: 4K pages, <BPO>; 32K pages, <BP32K>
TSPV01	V3	A table space assigned to the database DSNDB07 is invalid if one or all of the following parameters are specified: NUMPARTS FREEPAGE PCTFREE
TSPV02	V3	A nonsegmented table space is invalid if the value of the locking size is 'TABLE'.

Table 9. Rule Explanations (continued)

Rule ID	Version	Rule explanation
TSPV03	V3	A partitioned table space is invalid if the table assigned to the table space has no partitioning index.
TSPV04	V3	A table space is incomplete if there is no table assigned.
TSPV05	V3	A table space is incomplete if no volume identifiers are specified for the assigned storage group.

Appendix H. Performance Considerations

Certain DataAtlas operations, described below as “CPU-intensive,” may degrade the performance of other DataAtlas operations that would otherwise perform well. If many users will be active in DataAtlas concurrently, schedule CPU-intensive operations for a time when the expected user load will be light.

In each execution of DataAtlas, there is an initial overhead for the first access of each object type in the TeamConnection database. Later accesses do not incur this overhead.

If you run the DataAtlas client on a separate machine from your TeamConnection server, the performance of most DataAtlas tasks will improve.

With the TeamConnection server on its own machine, set the OS_CACHE_SIZE environment variable for the TeamConnection server daemons to total at least half the total memory of the machine, but leave at least 16MB available. For example, on a server machine with 64MB RAM with two server daemons, set OS_CACHE_SIZE for each daemon to at least 16MB (16MB*2 is half of 64MB) but no more than 24MB (24MB*2 is 48MB, which leaves 16MB). This will improve the performance of the largest DataAtlas tasks, such as populates and Modeler transforms.

Populating

Populating can be quite CPU-intensive, depending on the number of objects being populated. To obtain adequate performance during a populate operation, close any workfolders that contain objects used during the operation. This allows DataAtlas to clean up the local cache memory as the operation progresses.

Maintenance

Opening notebooks for certain DB2/390 objects also loads information for related objects into local memory. For example, opening an index loads information about the related table; opening a table loads other tables for which it has a foreign key. Retrieving this additional information slows the performance of opening the notebook. However, after the additional information is in local memory, the performance of opening the related objects improves.

Objects that are generated or viewed via their notebooks may take up space in local memory while a workfolder is open, and performance is degraded over time. Periodically close your workfolders to clean up local memory. This practice improves performance and saves the state of the workfolders. Also, close and restart DataAtlas each day for a complete local memory cleanup.

Queries

Queries that involve many DataAtlas objects or that span many objects can be very CPU-intensive. To get the best performance from your SQL queries, write queries that reference only the columns you are interested in. A few view types have hundreds of columns, so returning all the columns can be slow. For example:

1) SELECT d.* FROM OUTER DataAtlasDBD d; --> potentially long-running
2) SELECT d.rootDBDname FROM OUTER DataAtlasDBD d; --> faster query

Designer

These DataAtlas Designer activities are CPU-intensive:

- Concurrent propose, validate, or inform tasks on many objects
- The Propose Foreign Key task on a given table when the physical design contains many tables
- Opening a physical design group (DB2/390 table, DB2/390 table space, and so on) with more than 50 objects in the group

Modeler

DataAtlas Modeler"s forward and reverse transform operations can be very CPU-intensive. If a model has more than 100 entities, do these operations on a client machine with more than 32MB RAM.

Glossary

active window. The window currently in use. It receives keyboard input and is distinguishable by the unique color of its title bar and window borders.

build. The process used to create applications within TeamConnection.

build script. An executable or command file that specifies the steps that should occur during a TeamConnection build operation.

build tree. A graphical representation of the dependencies that the parts in a TeamConnection application have on one another. If you change the relationship of one part to another, the build tree changes accordingly.

data definition. Data that describes data, also known as "metadata." In relational database systems, tables, table spaces, indexes, views, storage groups, synonyms, and aliases have data definitions. In IMS, DBDs and PSBs have data definitions. In COBOL and PL/I, COPY files and include files have them.

data component. A data element or a data structure.

data element. The most elementary data type—for example, a field in an IMS segment or a column in a relational table.

data structure. A collection of data elements.

dot-dot operator. TeamConnection SQL notation that expresses linkage in queries.

generate. To produce a data definition for use in an operating environment by using object information in the TeamConnection database.

HLL. High-level language.

included source definition. An object that defines an included source file. For COBOL, an included source file is a COPY file; for PL/I, an include file.

local data component. A local data element or a local data structure.

local data element. The smallest unshared data unit within a data definition; for example, a relational table column, an IMS database field, or a COBOL or PL/I elementary item.

local data structure. An unshared data structure containing local data elements.

mapping table. A table that shows associations between local data components and shareable data components. See also *prototype mapping table*.

object data model. A data model in which data is represented as objects that have attributes and relationships between objects.

populate. To import a data definition into the TeamConnection database from an external source.

prototype mapping table. A mapping table DataAtlas produces when the mapping table being used has no entries for some of the local data components being processed. It's a prototype because the source names and target names in its entries are identical; you'll probably want to differentiate the names of your shareable data components.

query. A request for information from the TeamConnection database qualified by specified conditions.

reconcile. To replace newly populated data items with references to shareable objects.

relational data model. A data model with a pattern of organization based on a set of relations defined in the form of tables, in rows and columns.

shareable data component. a shareable data element or a shareable data structure.

shareable data element. A TeamConnection object that can be used repeatedly to define data items like relational table columns, IMS database fields, COBOL elementary items, and PL/I elementary items.

shareable data structure. A TeamConnection object that can be used repeatedly to define data structures like IMS segments, COBOL group items, and PL/I group items.

shareable table definition. A shareable object representing the basic definition of a relational table.

TeamConnection SQL. A superset of the SQL entry-level standard of 1992. TeamConnection SQL is based on an object data model, where data is represented as objects that have attributes and relationships between the objects.

version. A given family, release, and work area within TeamConnection.

workfolder. A place for collecting TeamConnection database objects that belong to the same version.

Index

A

accepting design proposals 92
access to tables 74
adding
  columns to a table 44
  data items to a data structure 26, 48
  data items to a shareable data structure 50
  fields to a segment 45
  PCBs to a PSB 46
  segments to a database 45
adding, data items to a data structure 22
adding columns 97
alias/synonym
  assigning to physical design 136
  defining 135
  designing 135
  specifying general information 135
assigning
  alias/synonym to physical design 136
  database to physical design 125
  database to storage group 125, 129
  index to physical design 112
  storage group to physical design 128
  table space to a database 119
  table space to a physical design 118
  table to physical design 98
  table to table space 103, 121
  view to physical design 134

B

buffer pool
  design of 77
  for database 125
  for table space 121
  in physical design 72
build script
  description 55
  editing 145
  sample 145
  using 145

C

COBOL
  build script settings 150
  creating objects 22
  cross-generation form PL/I 54
  generating COPY files 53
  populating COPY files 35
  sample command file 139
  sample files 138
column
  adding to a table 97, 44
  available design advice and support 107
  creating new 97
  data type 108
  defining 108
  deleting from a table 97, 45
  designing 107
  in physical design 72
  in relational design 71
  index 113
  setting options 109
  specifying data load 109
  specifying work load 110
  subtype 108
  viewing DB2 actuals 110
concurrent development 155
configuring design environment 83
converting IMS object code 33
converting used space value of storage group 130
creating
  COBOL objects 22
  columns 97
  DB2/390 objects 19
  DB2 UDB objects 19
  foreign key 101
  IMS objects 21
  index 102
  new database 80
  Oracle objects 19
  PL/I objects 22
  primary key 99
  prototype mapping tables 41
  relational database objects 19
  shareable data components without reconciling 39
  shareable data element 24
creating *(continued)*
  shareable data structures 24
  table partitions 103
  table space partitions 119
  unique key 100

D

data definition language (DDL) 69
data items
  adding to a data structure 26, 48
  adding to a shareable data structure 50
  deleting from a data structure 23, 49
  deleting from a shareable data structure 51
data items, adding to a data structure 22
data load
  design information 78
  for column 109
  for table 105
  overview 72
data type of column 108
DataAtlas
  Designer considerations 30
  Help information 8
  Main Folder window 7
  Modeler considerations 30
  Profile notebook 11
  tutorial 8
DataAtlas Dictionary 69
DataAtlas Modeler 69
database
  assigning to physical design 125
  assigning to storage group 125, 129
  available design support 123
  designing 123
  in physical design 72
  selecting buffer pool 125
  setting options 124
  specifying design information 124
  specifying general information 124

database design
concepts 71
knowledge on 74
physical 69, 74
process 69
relational 69, 74
starting from a notebook 86
starting from a physical design 84
with DataAtlas Designer 75
database objects
designing using notebooks 86
optimizing 80
physical 72
relational 71
database system 69
DB/DC Data Dictionary, migrating definitions from 5
DB2/390
generating 53
sample build script settings 147
DB2/390 sample build script settings 147
DB2/400, using DataAtlas DDL 55
DB2 actual values 78
DB2 UDB
generating 53
populating tables 31
sample build script settings 147
sample files 137
DB2 values
of column 110
of index 115
of storage group 131
of table 106
of table space 121
defining
alias/synonym 135
column 108
index 112
view 133
definitions, generating 53
deleting
columns from a table 45
data items from a data structure 23, 49
data items from a shareable data structure 51
fields from a segment 46
PCBs from a PSB 46
segments from a database 46
deleting columns 97
design
scenarios 79
design *(continued)*
using design support 75
using notebooks 75
design actions
selecting 88
tailoring 88
design areas 77
design environment, configuring 83
design information
overview 78
specifying for database 124
specifying for table space 121
design modes 75
design proposals
accepting 92
executing 93
selecting from a choice of 93
design report
evaluating 90
requesting 90
design reports 79
design support
evaluating report 90
for column 107
for database 123
for index 111
for storage group 127
for table 96
for table space 117
overview 75
report 79
requesting from a notebook 88
requesting from a physical design 84
requesting from a workfolder 85
requesting report 90
types of 75
Designer, considerations when populating 30
designing
alias/synonym 135
column 107
database 123
index 111
set of objects 78
single objects 77
storage group 127
table 95
table space 117
view 133
disassociating shareable data elements
from a column 45
from a data structure 23, 49
disassociating shareable data elements *(continued)*
from a field in a segment 46
from a shareable data structure 51

E

engineering
forward 69
reverse 69
existing database, optimizing 80
extracting DB2 values
for table 106
for table space 121

F

fields
adding to a segment 45
deleting from a segment 46
foreign key in relational design 71
forward engineering 69

G

general information
for alias/synonym 135
for database 124
for index 112
for storage group 128
for table 96
for table space 118
for view 133
generating
data definitions from DataAtlas 53
description text for DataGuide 53
using datatlas.exe build script 145

H

help information 8
HLL cross-generation 54

I

IMS build script settings 150
IMS DBD
populating 33
updating 45
IMS name qualifiers 143
IMS objects
converting to source code 33
creating 21
generating 53
IMS PSB
populating 33

IMS PSB *(continued)*
updating 46
IMS sample build script settings 150
IMS sample files 138
included source definition 35
index
assigning to physical design 112
available design advice and support 111
columns 113
creating 102
defining 112
designing 111
in physical design 72
of storage group 129
sorting order 113
specifying general information 112
storage 114
viewing DB2 values 115
viewing partitions 114
index partition
storage 114
type 114

K

key
foreign 101
primary 99
unique 100

L

launching DataAtlas 9
locking considerations 154

M

Main Folder window 7
mapping table
examples of entries 41
prototype 41
structure 39
migrating data definitions 5
model
physical 69
relational 69
Modeler, considerations when populating 30
modifying
foreign key 101
primary key 99
table partitions 104
unique key 100

N

naming
conventions 12
examples 13
IMS name qualifiers 143
object name qualifiers 141
relational name qualifiers 142
notebooks
design with 75
requesting design support from 88

O

object
creating 19
deleting 27
renaming 153
searching for 15
updating 43
object name qualifiers
IMS qualifiers 143
relational qualifiers 142
online information
DataAtlas help 8
DataAtlas tutorial 8
optimizing
existing database 80
single database objects 80
table layout 98
optimizing access path 102
options
for column 109
for database 124
for table 99
for table space 119
Oracle, sample build script settings 149
Oracle objects, generating 53

P

PCBs
adding to a PSB 46
deleting from a PSB 46
performance considerations 225
physical design
assigning aliases/synonyms to 136
assigning databases to 125
assigning indexes to 112
assigning storage groups to 128
assigning table spaces to 118
assigning tables to 98
assigning views to 134
requesting design support from 84
physical designs, generating 53
PL/I
build script settings 151
creating objects 22
cross-generation form COBOL 54
generating include files 53
populating include files 35
sample command file 139
sample files 139
supported data and nondata attributes 157
populating
COBOL COPY file 35
DB2 UDB tables 31
Designer and Modeler considerations 30
from DB2 UDB catalogs 31
from workstation directories 33
IMS DBD 33
PL/I include file 35
primary key in relational design 71
Profile notebook
overview 11
version information 12
prototype mapping tables 41

Q

qualifiers
IMS object name 143
prefix 144
relational object name 142
queries
running supplied queries 58
running your own 57
supplied queries 57
writing TeamConnection SQL 61

R

reconcile mapping table samples 139
reconciling similar objects 38
relational database objects, creating 19
relational name qualifiers 142
renaming objects 153
report, design support 90
required space, of storage group 130
reusing objects 38
reverse engineering 69
routines for table 99

rules
basis for design support 75
explanations 214
IDs 211
modifying 77
tailoring the set 76

S

sample build script settings
for COBOL objects 150
for DB2 objects 147
for IMS objects 150
for Oracle objects 149
for PL/I objects 151
sample files
COBOL command file 139
COBOL COPY files 138
DB2 UDB 137
IMS 138
PL/I command file 139
PL/I include files 139
reconcile mapping tables 139
scenarios 79
searching
for TeamConnection database objects 15
from a workfolder 15
segments
adding to a database 45
deleting from a database 46
selecting
buffer pool for database 125
buffer pool for table space 121
from a choice of proposals 93
serial development 155
setting
options for column 109
options for database 124
table space options 119
shareable data component 38
creating without reconciling 39
shareable data element 97, 108
shareable data elements
creating 24
disassociating from a column 45
disassociating from a data structure 23, 49
value of 24
shareable data structures
creating 24
value of 24
shareable table definition 97
sorting order of index 113
specifying
data load for column 109
data load for table 105
design information for database 124
design information for table space 121
index columns 113
required space of storage group 130
sorting order of index 113
storage information of storage group 128
storage of index 114
storage of index partition 114
storage of table space 120
storage of table space partitions 120
table routines and options 99
type of index partition 114
type of table space partitions 120
usage intent of storage group 130
work load for column 110
work load for table 105
storage group
assigning database 129
assigning to physical design 128
available design support 127
converting used space value 130
designing 127
in physical design 72
specifying general information 128
specifying required space 130
specifying storage information 128
specifying usage intent 130
viewing DB2 values 131
storage information, for storage group 128
storage of index 114
storage of table space 120
storage of table space partition 120
storage of tables 73

T

table 71
assigning to physical design 98
assigning to table space 103, 121
available design advice and support 96
creating foreign key 101
creating partitions 103
table *(continued)*
creating primary key 99
creating unique key 100
designing 95
extracting DB2 values 106
in physical design 72
in relational design 71
layout 98
modifying foreign key 101
modifying partitions 104
modifying primary key 99
modifying unique key 100
optimizing layout 98
specifying data load 105
specifying general information 96
specifying routines and options 99
specifying work load 105
viewing shareable definition 97
viewing the columns 97
table partition
creating 103
modifying 104
table space
assigning table 121
assigning tables 103
assigning to a database 119
assigning to a physical design 118
available design support 117
creating partitions 119
designing 117
extracting DB2 values 121
in physical design 72
of database 124
of storage group 129
selecting buffer pool 121
setting options 119
specifying design information 121
specifying general information 118
table space partition
creating 119
storage 120
type 120
tables, relationship between 69
TeamConnection
build script 55
cache and workfolders 153
concurrent development 155
considerations 153
locking 154

TeamConnection *(continued)*
serial development 155
setup overview 10
TeamConnection SQL queries 61
trial run
making a prototype mapping table 41
running 37
tutorial for DataAtlas 8

U

unique key in relational design 71
updating
COBOL included source definitions 47
DB2/390 definitions 43
DB2 UDB definitions 43
IMS DBD objects 45
PL/I included source definitions 47
relational database objects 43
shareable data structures 49
usage intent, of storage group 130
used space value, of storage group 130

V

version information in Profile notebook 12
view
assigning to physical design 134
defining 133
designing 133
in relational design 71
specifying general information 133
viewing
DB2 actuals of column 110
DB2 actuals of index 115
DB2 actuals of storage group 131
index partitions 114
indexes of storage group 129
shareable table definition 97
table spaces of database 124
table spaces of storage group 129
the table columns 97
VisualAge Exchange, using to migrate definitions 5
volume 77

W

work load
design information 78
work load *(continued)*
for column 110
for table 105
overview 72
workfolder
and TeamConnection cache 153
creating 15
naming 11
relation to TeamConnection 11
requesting design support from 85
searching from 15
specifying version information for 12

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC26-9134-00