DB2 Data Management Software

IBM

Moving Data Across the DB2 Family

- Explore the usability and performance of the DB2 Family's functions
- Try out the High Performance Unload on several platforms
- Experience the new Load from Cursor utility option

Paolo Bruni
Michael Ho
Marko Milek
Arne Nilsen
Jose Mari Michael Sanchez

Redbooks

ibm.com/redbooks

International Technical Support Organization

Moving Data Across the DB2 Family

February 2003

SG24-6905-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xvii.

First Edition (February 2003)

This edition applies to DB2 UDB for z/OS Version 7 (Program Number 5675-DB2), DB2 Version 7 Utilities Suite (Program Number 5697-E98), DB2 UDB V7.2 for Linux, UNIX, and Windows (Program Number 5648-D37) and the pre-GA (beta) DB2 UDB V8.1 for Linux, UNIX, and Windows (Program Number 5765-F34), High Performance Unload for z/OS Version 2 Release 1 (Program Number 5655-I19) and High Performance Unload for Multiplatforms Version 2 Release 1 Modification 2 and 3 (Program Number 5724-B90.)

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

Figures

Examples

Tables

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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

AIX®
AS/400®
CICS®
COBOL/2™
DataJoiner®
DB2®
DB2 Connect™
DB2 Universal Database™
DFS™
DFSMSdfp™
DFSMSdss™
DFSMShsm™
DFSORT™
Distributed Relational Database Architecture™
DRDA®
eServer™
Enterprise Storage Server™
IBM®
IMS™
Infoprint®
Informix®
IP PrintWay™
iSeries™
Lotus®
MQSeries®
MVS™
NetView®
Notes®
NUMA-Q®
OS/2®
OS/390®
OS/400®
Perform™
PrintWay™
PTX®
QMF™
QBIC®
RACF®
RAMAC®
Redbooks™
RETAIN®
S/390®
S/390®
Sequent®
SOM®
SQL/DS™
System/390®
Tivoli®
TME®
WebSphere®
Word Pro®
z/OS™
zSeries™
1-2-3®Redbooks(logo)™ 

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

Moving data across different databases and even different platforms has been a common task in IT shops for quite some time. Applications may have been developed independently and over time, using packages and different technology; and data might reside on different platforms exploiting the specific platform strong points. However, there still is a growing need for applications that need to access all of this data for overall processing.

While new Web related technologies are emerging with the intent to provide functions to collect and integrate information across multiple databases and applications for access in real time, moving data to one location for overall processing is still a very common requirement.

This IBM Redbook provides an overview of what is currently available within the DB2 Family of products (specifically DB2 for z/OS, and DB2 for UNIX and Windows) in terms of functions, tools, and utilities to satisfy the need for moving data. We focus on discussing High Performance Unload and Cross Loader; the first one is a tool, the second one is a new option of the Load utility, since they are the latest functions that IBM has released. We also introduce the concepts and some examples of using the Federated Database support.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Paolo Bruni has been a Project Leader since 1998 at the International Technical Support Organization, San Jose Center, where he conducts projects on all areas of Data Management. During his many years with IBM, in development and in the field, Paolo's work has been mainly related to database systems.

Michael Ho is a Software Engineer at the IBM Toronto Laboratory in Canada. He has four years of experience in database development. Michael holds a BS degree from the University of Toronto. His areas of expertise include the Load, Import, and Export utilities for DB2 UDB on distributed platforms.

Marko Milek is a Software Engineer at the IBM Toronto Laboratory in Canada. He has two years of experience in database development. Marko holds a bachelor's degree from the California Institute of Technology, and a Ph.D. from McGill University. His areas of expertise include utilities and tools for DB2 UDB on distributed platforms.

Arne Nilsen is a Database Administrator in ErgoSolutions in Oslo, Norway. He has more than 15 years of experience in data management. Arne has an extensive knowledge in most areas of DB2 gained through his wide experience in the field. During the last several years he has been involved in system design, integration, and security.

Jose Mari Michael Sanchez is an IT Specialist for IBM Global Services in the Philippines. He has 6 years of experience in the data management field. He started as an application developer and then became a DB2 technical support and system programmer for OS/390. Michael holds a BS in Applied Physics from the University of the Philippines. He is an IBM Certified Solution Expert for DB2 UDB V7 Database Administration.

Thanks to the following people for their contributions to this project:

Nagraj Alur
Corinne Baragoin
Emma Jacobs
Bart Steegmans
Maritza Marie Dubec
International Technical Support Organization, San Jose Center

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Dinesh Nirmal
Jim Ruddy
Dave Schwartz
Randy Spalten
Don Weil
IBM Silicon Valley Laboratory

Mark Leitch
IBM Toronto Laboratory

Norbert Thiebaud
Infotel

Become a published author

Join us for a two- to seven-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2

650 Harry Road
San Jose, California 95120-6099

Part 1

Introduction

In this part we introduce the possible reasons for moving data across the DB2 subsystems and the different platforms, the objectives of our project, and the contents of this redbook, as well as provide a summary of the functions and tools that can help in this area.

This part is structured in the following two chapters:

- Introduction
- Overview of data movers

1

Introduction

Moving data across different databases and different platforms has been a common task in IT shops for quite some time. By *moving data* we mainly refer to the task of copying data from a source system to a target system, not necessarily removing it at the source location. While relational databases, and especially the DB2 Family of products, might have the same look and feel, each platform has its own strengths and advantages. So, data in a typical database shop will most probably reside in different platforms. There are also many instances of companies merging for business reasons, each one with an IT shop with its own characteristics, where the applications need to access all of this data even if they reside in different databases on different platforms. Currently, you can then either process them remotely, which is a valid solution for infrequent processing and reasonable amounts of data, or you need to move the data to the most suitable place for the overall processing.

IBM has been at the forefront of Data Management for a long time and is now starting to offer solutions in the area of information integration. Information integration is a collection of technologies comprising DBMS, Web services, replication, federated systems, warehouse functions, programming interfaces, and data models into a common platform, which provides an end-to-end solution for transparently managing the high volume and the diversity of today's enterprise data. A good description of the foundation for this effort, which is likely to greatly reduce the need for moving data, is reported in a series of articles contained in the recent issue of the *IBM Systems Journal - Vol. 41, No. 4, 2002,* G321-0147.

In this IBM Redbook we concentrate on the instances where DB2 data has to be moved, or copied, sporadically or periodically, across different database instances and different platforms.

Each member of the DB2 Family (DB2 UDB for z/OS, DB2 UDB for Linux, UNIX and Windows, and DB2 for iSeries) supports several functions and utilities to move data across and within platforms.

Starting in the year 2001, IBM has also been delivering new tools. They are new competitive products, or function-rich new versions, which provide tool integration for your DB2 and IMS installations. These tools will:

- Maximize the availability of your systems
- Keep your operating environment running at peak performance
- Meet requirements for additional recovery and replication capabilities

- Effectively create and operate your application subsystems

Besides focussing on the database tools to complement your DB2 UDB for z/OS and IMS database engines, IBM is now also providing tools for the distributed platforms by releasing new tools and versions at a fast rate.

This book provides an overview of what is currently available for DB2 for z/OS and DB2 for Linux, UNIX, and Windows in terms of functions, utilities, and tools. We also discuss in detail the two latest functions and interesting functionalities in the area of moving data: the High Performance Unload and the Cross Loader. The first one is a tool, the second one is a new option of the Load utility.

Since it is unrealistic to include all possible permutations of functions, tools, and environments, we have made some choices and defined some boundaries for our activities based on what we considered more critical or what is not already documented.

1.0.1 Platforms and configurations

We have classified the platforms into two environments as depicted in Figure 1-1:

- **The mainframe platform**

  The mainframe environment is represented by:

  – z/OS or OS/390 on zSeries or S/390

  In our project for the mainframe environment, we have used DB2 for z/OS and OS/390 Version 7, on a zSeries with z/OS V1R3.

- **The distributed platform**

  The distributed environment is represented by Intel-based and the RISC-based operating systems which include:

  – Window NT or 2000 and Linux
  – AIX, SunOS, and HP

  In our project for the distributed environment, the examples shown in this redbook were tested on AIX and Windows. Since DB2 UDB is a product with very consistent capabilities across all distributed platforms, the functionalities are expected to be equivalent for all systems mentioned. In our project we have used DB2 UDB for UNIX and Windows Version 7.2, and a beta level of Version 8. Version 8 was required for the DB2 Cross Loader (or Load from cursor) function.

In terms of permutations on moving data, we have considered the cases of moving data within the same environment, mainframe or distributed, and across the two platforms.

The objectives of this project are to:

- Try out the different functions available to DB2 UDB for moving data. We explore the strengths and advantages of each when moving data.
- Show the circumstances where each tool or utility is most suitable for the need.
- Present a comparative analysis in terms of performance and functions whenever possible.

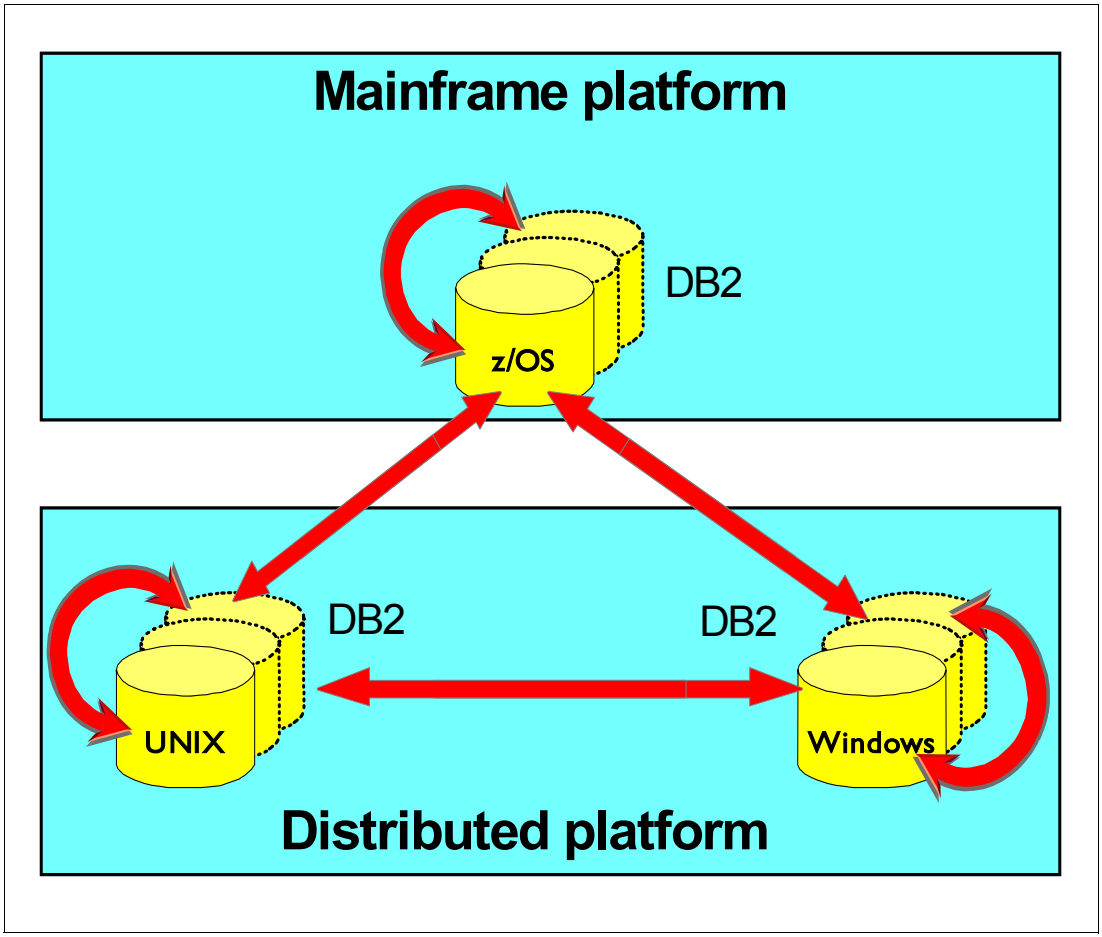

Figure 1-1 The platforms used in this project

1.0.2 Terminology

Throughout this redbook we have tried to be meaningful and consistent with the terminology related to the DB2 Family of products, and the platforms where they run, but every generalization and abbreviation tends to be arbitrary and sometimes wrong, so we summarize here our definitions for reference purposes.

DB2 for z/OS means DB2 UDB for z/OS and OS/390. All our tests were at DB2 UDB for z/OS and OS/390 Version 7 level.

DB2 distributed means DB2 UDB for Linux, UNIX, and Windows. Our tests were done with DB2 UDB Version 7.2 and 8 on AIX and Windows. The term *DB2 UDB* is commonly used as a way of differentiating DB2 distributed from DB2 for OS/390, but it is incorrect since DB2 for OS/390 also became UDB with Version 6.

Mainframe platform is the zSeries with z/OS platform.

Distributed platform is any UNIX or Intel based platform (Windows or Linux).

A few times *multiplatform* has been used interchangeably with distributed platform (like in the name of products), while *crossplatform* has been used to include both the mainframe and the distributed environments.

1.1 Contents of this redbook

These are the parts and chapters contained in this book:

- **Part 1: Introduction**

  We briefly discuss the different DB2 data movement functions and utilities in the first part of this book. We then present an overview of the capabilities and functions of most IBM tools available for this purpose.

- **Part 2: Product functions and utilities**

  In the second part, we analyze in more detail the functionalities of the DB2 subsystems and related utilities. We present the advantages and disadvantages of the following functions:

  - Unload with DB2 for z/OS
  - Load with DB2 for z/OS
  - Export and Import with DB2 distributed
  - Load with DB2 Distributed

- **Part 3: High Performance Unload**

  In this part we examine functions and performance of the recent new releases of the two flavors of the IBM High Performance Unload tool:

  - High Performance Unload for z/OS
  - High Performance Unload for Multiplatforms

- **Part 4: Scenarios**

  In this part we consider how to become prepared to execute sample scenarios of moving data within and across the host and distributed platforms:

  - Getting ready for moving data
  - Moving data to DB2 for z/OS
  - Moving data to DB2 Distributed

2

Overview of data movers

In this chapter we provide an overview and a summary of the functions, utilities, and tools that can be considered useful when dealing with the task of moving data.

We have associated these data movers to three areas as follows:

- DB2 UDB for z/OS:
  - DB2 DSN1COPY utility
  - DB2 sample program DSNTIAUL
  - DB2 Reorg utility
  - DB2 Unload utility
  - DB2 Load utility
  - DB2 Cross Loader option
- DB2 UDB for UNIX and Windows:
  - DB2 Backup and Restore utilities
  - DB2 Export utility
  - DB2 Import utility
  - DB2 Load utility
  - DB2 Cross Loader option
  - DB2 db2move
  - DB2 db2look
- Tools for z/OS and multiplatforms:
  - DB2 Administration Tool for z/OS
  - DB2 Data Export Facility tool for z/OS
  - DB2 High Performance Unload tool for z/OS
  - DB2 Data Replication tools for z/OS and Multiplatform
  - DB2 Web Query tool for z/OS and Multiplatform
  - DB2 High Performance Unload tool for Multiplatforms
  - DB2 UDB Warehouse Manager for UNIX and Windows

2.1 Preliminary considerations

There are some basic concepts to consider and decisions to be made before you start moving data:

- Verify that you are copying only clean, consistent, and well defined data.

  Before copying any DB2 data, you must resolve all situations where data is in an inconsistent state. On DB2 for z/OS use the DISPLAY DATABASE command to determine whether any inconsistent state exists, and the RECOVER INDOUBT command or the RECOVER utility to resolve the inconsistency.

  Furthermore, copying or moving the data is not enough. These operations have as a prerequisite the copying of DB2 object definitions. And when copying from one DB2 for z/OS subsystem to another, you must consider internal values that appear in the DB2 catalog and the log; for example, the DB2 object identifiers (OBIDs) and log relative byte addresses (RBAs). If you are copying across different platforms you must verify and possibly adjust the data definition language statements to best fit with the characteristics of the platform you are moving to.

- Be careful when copying DB2 data with non-DB2 functions.

  Although DB2 for z/OS data sets are created using VSAM access method services, they are specially formatted for DB2 and cannot be processed by standard VSAM services that use record processing. They can be processed by VSAM utilities that use control-interval (CI) processing, and since they are linear data sets (LDSs), also processed by utilities that recognize the LDS type.

  You can use DFSMSdss to copy data between disk devices. In this redbook we do not cover functional components in DFSMS that merely copy data set like the Data Set Services (DFSMSdss).

  Data Facility Product (DFSMSdfp) is a prerequisite for DB2. You can use access method services EXPORT and IMPORT commands with DB2 data sets when control interval processing (CIMODE) is used.

  Hierarchical Storage Manager (DFSMShsm) provides support for the MIGRATE, HMIGRATE, or HRECALL commands. By specifying the data set names, you can move data sets from one disk device type to another within the same DB2 subsystem.

  For more details, see *DB2 UDB for OS/390 and z/OS V7 Administration Guide*, SC26-9931-03.

  Similar considerations are applicable to DB2 distributed.

- More preliminary considerations are listed in Chapter 9, “Getting ready for moving data” on page 213.

2.2 DB2 UDB for z/OS

In this section we provide an overview of the functions and utilities available for moving data with DB2 UDB for z/OS.

2.2.1 DB2 DSN1COPY utility

DSN1COPY is a DB2 for z/OS stand-alone utility that does not need DB2 active to execute. It provides function for copying table spaces from the DB2 VSAM data sets or from the DB2 image copy data sets.

All tables in the table space will be copied.

You can translate database object identifiers (OBIDs) to enable moving data sets between objects with different object identifiers in the same or between different subsystems and reset the log RBAs in the target data set.

The following steps should be followed:

1. Start the table space as read-only.
2. Run the QUIESCE utility with the WRITE (YES) option to externalize all data pages and index pages.

DSN1COPY does not require DB2 authorization. However, usually the data set is protected by Resource Access Control Facility (RACF); if so, you need sufficient RACF authorization.

For more details, see: *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03.

DSN1COPY should only be used when you need to access the data outside DB2 or you need to do an OBID translation. All other needs can be solved with utilities and tools mentioned in the following sections.

2.2.2 DB2 sample program DSNTIAUL

DSNTIAUL is a dynamic sample SQL unload program included in the DB2 for z/OS product as a sample program.

DSNTIAUL unloads some or all rows from up to 100 DB2 tables. With DSNTIAUL, you can unload data of any DB2 built-in data type or distinct type. DSNTIAUL unloads the rows in a form that is compatible with the Load utility and generates control statements for the Load utility.

DSNTIAUL is written in assembler language and is shipped only as source code, so you must precompile, assemble, link, and bind it before you can use it.

If you choose to specify the SQL parameter, your input must contain one or more complete SQL statements. Each of them must end with a semi-colon. You can include any SQL select statement that can be executed dynamically in your input data set.

If you do not specify the SQL parameter, your input data set must contain one or more single-line statements (without a semi-colon) that use the following syntax:

```
table or view name [WHERE conditions] [ORDER BY columns]
```

Each input statement must be a valid SQL SELECT statement with the clause SELECT * FROM omitted and with no ending semi-colon. DSNTIAUL generates a SELECT statement by appending SELECT * FROM for each input statement. For this input format, the text for each table specification can be a maximum of 72 bytes and must not span multiple lines.

DSNTIAUL offers two types of output:

- The result set from the SQL will be written to the output file (SYSRECnn) specified.
- The Load utility control statements for loading rows into a corresponding table will be written to the output file (SYSPUNCH) specified.

For more details, see *DB2 UDB for OS/390 and z/OS V7 Administration Guide*, SC26-9931-03, and *DB2 UDB for OS/390 and z/OS Version 7 Application Programming and SQL Guide,* SC26-9933.

For years DSNTIAUL has been an obvious choice for unloading data or for generating LOAD control statements. You have to use the DB2 Load utility for loading the data into an existing table.

DSNTIAUL cannot unload data from an image copy, only from a table.

The SQL parameter gives you the possibility to include delimiters in the result set through constants in the select list.

Currently, there are several alternatives to DSNTIAUL, each one with its characteristics described in more detail in the corresponding sections:

- DB2 Reorg utility with the UNLOAD EXTERNAL option provides faster unloading than the DSNTIAUL, but it does not have the SQL option and it requires Reorg utility authorization.
- DB2 Unload utility provides faster unloading than the DSNTIAUL, but it does not have the SQL option. The authorization requires SELECT authorization, like DSNIAUL.
- DB2 High Performance Unload tool provides performance similar to or better than the Unload utility and SQL flexibility. It offers the DSNTIAUL format option on output. If you want better performance than with your old DSNTIAUL solutions, you should consider this tool.

2.2.3 DB2 Reorg utility

With DB2 for z/OS V6 the Reorg utility has been enhanced with an unload capability activated through the REORG UNLOAD EXTERNAL optional statement.

This option results in a format usable by the Load utility. It also generates utility control statements for the Load if requested.

With use of the WHEN clause you can also restrict the data being unloaded.

Reorg authorization is needed to run the Reorg utility.

For more details, see *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03.

The REORG UNLOAD EXTERNAL is an alternative or a supplement to DSNTIAUL. It can only unload data from a table, not an image copy. You have to use the DB2 Load utility for loading the data into an existing table.

DB2 V7 offers a new utility, DB2 Unload, which can be considered a replacement to REORG UNLOAD EXTERNAL. DB2 Unload provides faster unloading than the Reorg utility and only the SELECT authorization is needed

2.2.4 DB2 Unload utility

The Unload utility, introduced with DB2 for z/OS V7, can be used to unload DB2 data to sequential data sets.

Unload can be used to unload data from one or more source objects to one or more sequential data sets in external formats. The sources can be:

- DB2 table spaces
- DB2 Image Copy data sets, including inline copies taken during Reorg and Load utility. The table space of which the image copy is being unloaded must exist. Image Copies of dropped table spaces cannot be used.

Major options available with Unload are:

- Unload of data from DB2 tables with SHRLEVEL CHANGE, which ensures availability
- Conversion between formats as EBCDIC, ASCII, UNICODE and CCSID
- Sampling, limiting of rows
- Partition parallelism

The Unload utility requires SELECT authorization on the table or tables in the table space.

For more details, see Chapter 3, “Unload with DB2 for z/OS” on page 33, and *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03. DB2 V7 offers new possibilities. The recently announced DB2 for z/OS Version 8 offers the possibility of unloading data into a delimited file. See *DB2 UDB for z/OS Version 8 What's New?*, available from the Web site:
http://www.ibm.com/software/data/db2/os390/db2zosv8.html

For performance considerations see the Redbooks *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite,* SG24-6289, and *DB2 UDB for OS/390 and z/OS Performance Topics,* SG24-6129.

You should consider DB2 High Performance Unload tool as alternative to the DB2 Unload utility:

- Performance tests shows a slightly better performance than the Unload utility, and considerable less CPU time in execution. To gain these figures it is an absolute condition that it does not use the DB2 engine to perform the SQL.
- It provides you with more versatility because you can use the SQL Select statement to select the rows and columns that you want to be unloaded.
- If you want to unload from an incremental image copy
- It also provides you with more flexibility regarding customizing the output through the USER DEFINED format.
- The DELIMTED format is compatible with the DEL format on a distributed platform.

2.2.5 DB2 Load utility

The Load utility populates DB2 tables with data from a sequential data set. The data can also be loaded from a user defined cursor with the new capability, called Cross Loader, which has been introduced with DB2 for z/OS Version 7. Input to the Load can therefore be a:

- Sequential data set
- Result set from SQL query against a DB2 table or view (see 2.2.6, “DB2 Cross Loader option” on page 12)

With the Load utility you can:

- Load rows into an empty table
- Add new rows into a table that is not empty
- Empty a table space before loading the table(s)
- Build or extend any index defined on the table
- Convert between encoding schemes as EBCDIC, ASCII, UNICODE, and CCSID
- Filter the input data
- Discard input data that does satisfy defined uniqueness, referential integrity or other constraints

Other functions worth mentioning:

- Inline COPY of the table being loaded

- Inline RUNSTATS of the table being loaded
- ONLINE RESUME option to ensure availability

If you do not have the ownership of the table to be loaded, you need Load authorization on the table. You also need SELECT authorization on the DB2 table or view if you are reading the result set from. The recently announced DB2 for z/OS Version 8 offers the possibility of loading data from a delimited file. See *DB2 UDB for z/OS Version 8 What's New?*, available from the Web site:
http://www.ibm.com/software/data/db2/os390/db2zosv8.html

For more details, see Chapter 4, "Load with DB2 for z/OS" on page 51 and *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03.

In regards to the performance of loading, the Load utility is currently superior to all other functions and tools.

2.2.6 DB2 Cross Loader option

The Cross Loader is not a new separate utility, but a new statement or option for the Load utility. It has a corresponding and equivalent function in the new cursor load capability of the Load utility of the DB2 for Linux, UNIX, and Windows Version 8. This option allows you to transfer data from one location for loading a DB2 table to another location within a single utility job.

Cross loading is done using the new DB2 V7 option **INCURSOR** in the DB2 Load utility. With this option you tell the Load to read the data from the result set of a cursor instead of reading it from an input sequential data set.

The data is read from the source location with a dynamic SQL statement (enclosed between EXEC SQL and ENDEXEC statements) and loaded into a table at the target location by the Load utility. The source data can be on the local system, on a remote DRDA server. The input source can also be on any system accessible via a *Federated Database* (see Appendix A, "Defining a Federated Database" on page 299.)

EXEC SQL is a new DB2 V7 utility statement that can be placed anywhere in the utility input stream. It can be used for two purposes:

- Executing a non-select dynamic SQL statement before, between or after the actual utility statements
- Declaring a cursor with a SQL select statement for use with the Load utility (Cross Loader). The declare cursor produces a result set.

> **Note:** EXEC SQL can simplify the JCL coding by eliminating dynamic SQL applications like DSNTIAD or DSNTEP2 from the JCL stream. It can merge different utility steps, separated by dynamic SQL applications, into one single utility step.

A typical Cross Loader example therefore consists of the definition of the dynamic SQL statement via the EXEC SQL DECLARE CURSOR utility statement, followed by a LOAD utility statement referring to this cursor. This is illustrated in Example 2-1 where we load an existing summary table called EMPSUMMARY with data coming from the local sample table DSN8710.EMP. The aggregation is done in the SQL statement of the CURSOR definition.

Example 2-1 Simple cross loading

```
EXEC SQL
DECLARE C1 CURSOR FOR
```

```
SELECT JOB,MAX(SALARY)AS MAX_SAL,MIN(SALARY)AS MIN_SAL
FROM DSN8710.EMP
GROUP BY JOB
ENDEXEC

LOAD DATA REPLACE
INCURSOR C1
INTO TABLE EMPSUMMARY
```

As you can see, it is a single job process that replaces the typical sequence of jobs of unloading, file transfer, and loading the data, locally or remotely.

For more details, see Chapter 4, “Load with DB2 for z/OS” on page 51 and *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03.

The Cross Loader with its EXEC SQL statement, is a very flexible and handy function of the Load utility. It combines the flexibility of the SQL statement and the performance of the Load utility. The source can either be on the mainframe, in a distributed system, or in any system accessible via a *Federated Database*.

2.3 DB2 UDB for UNIX and Windows

In this section we look at product functions, commands, and utilities for DB2 for UNIX and Windows.

2.3.1 DB2 Backup and Restore utilities

The DB2 UDB Backup and Restore utilities allow you to create a backup copy and then restore a table space or database from that copy within the UNIX, Windows, and Linux platforms, as well as converting from 32 bits to 64 bits. In more detail:

- **BACKUP DATABASE**

  It creates a backup copy of a database or a table space.

- **RESTORE DATABASE**

  It restores a possibly damaged or corrupted database that had previously been backed up using the DB2 backup utility. The restored database is in the same state it was in, when the backup copy was made. You have to consider ROLL FORWARD PENDING situations where you want to apply some of the log.

  This utility can also use the backup to create a new copy of the database. You can give a new name to the database. This has to be on the same platform where the backup was taken.

Either SYSADM, SYSCTRL or SYSMAINT authorization is needed when using backup or restore commands.

With reference to moving data, the DB2 UDB Backup and Restore utilities greatly facilitate the way to clone a table space or database within the same platform.

For details, see *IBM DB2 UDB Command Reference Version 8*, SC09-2951-01.

2.3.2 DB2 Export utility

The DB2 Export utility is used to extract data from a DB2 database. The exported data can then be imported or loaded into another DB2 database, using the DB2 Import or the DB2 Load utility.

The Export utility exports data from a database to an operating system file or named pipe, which can be in one of several external file formats. This file with the extracted data can be moved to a different server.

The following information is required when exporting data:

- An SQL SELECT statement specifying the data to be exported.
- The path and name of the operating system file that will store the exported data.
- The format (IXF, DEL or WSF) of the data in the input file.

The IXF file format results in an extract file consiting of both metadata and data. The source table (including its indexes) can be recreated in the target environment if the CREATE mode of the Import utility is specified. The recreation can only be done if the query supplied to the Export utility is a simple **`SELECT *`**

The following is an IXF example of the export command specifying a message file and the select statement:

```
export to stafftab.ixf of ixf messages expstaffmsgs.txt select * from staff
```

At least SELECT authorization is needed on the tables you export from.

The Export utility can be invoked through:

- The command line processor (CLP)
- The Export notebook in the Control Centre
- An application programming interface (API)

See Chapter 5., “Export and Import with DB2 distributed” on page 79 for restrictions that apply to the Export utility. For details, see Chapter 1 of *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830.

The Export utility can be used to unload data to a file or a named pipe from a table residing in the following:

- Distributed database
- Mainframe database through DB2 Connect (only IXF format)
- Nickname representing a remote source table (see Appendix A, “Defining a Federated Database” on page 299, for details).

Note: If performance is an issue, the DEL format covers your needs, and all rows are to be unloaded, then you should consider High Performance Unload tool for Multiplatforms. See 2.4.6, “DB2 High Performance Unload tool for Multiplatforms” on page 26 for a brief description. The tool must be executed from the machine where the source table resides.

2.3.3 DB2 db2batch

Exporting data in parallel into a partitioned database reduces data transfer execution time, and distributes the writing of the result set, as well as the generation of the formatted output, across nodes in a more effective manner than would otherwise be the case. When data is exported in parallel (by invoking multiple export operations, one for each partition of a table) it

is extracted, converted on the local nodes, and then written to the local file system. In contrast, when exporting data serially (exporting through a single invocation of the Export utility) it is extracted in parallel and then shipped to the client, where a single process performs conversion and writes the result set to a local file system.

The **db2batch** command is used to monitor the performance characteristics and execution duration of SQL statements. This utility also has a parallel export function in partitioned database environments that:

- Runs queries to define the data to be exported
- On each partition, creates a file containing the exported data that resides on that partition

A query is ran in parallel on each partition to retrieve the data on that partition. In the case of **db2batch -p s**, the original SELECT query is run in parallel. In the case of **db2batch -p t** and **db2batch -p d,** a staging table is loaded with the export data, using the specified query, and a **SELECT *** query is run on the staging table in parallel on each partition to export the data. To export only the data that resides on a given partition, **db2batch** adds the predicate NODENUMBER(*colname*) = CURRENT NODE to the WHERE clause of the query that is run on that partition. The *colname* parameter must be set to the qualified or the unqualified name of a table column. The first column name in the original query is used to set this parameter.

It is important to understand that **db2batch** runs an SQL query and sends the output to the target file, it does not use the Export utility. The Export utility options are not applicable to parallel export. You cannot export LOB columns using the **db2batch** command.

Run **db2batch -h** from the command window to see a complete description of command options.

The **db2batch** command executes a parallel SQL query and sends the output to a specified file. Note that the command is executing a select statement, not the Export utility. LOB columns, regardless of data length, cannot be exported using this method.

To export contents of the staff table in parallel:

```
db2batch -p s -d sample -f staff.batch -r /home/userid/staff.asc -q on
```

In this example:

- The query is ran in parallel on a single table (**-p s** option)
- Connection is made to the sample database (**-d sample** option)
- The control file staff.batch contains the SQL select statement (**select * from staff**)
- Output is stored to staff.asc file, default output format is positional ASCII (remember that db2batch is not using the Export utility)
- Only the output of the query will be sent to the file (**-q on** option)

To export into a delimited ASCII file:

```
db2batch -p s -d sample -f emp_resume.batch -r /home/userid/emp_resume.del,
/home/mmilek/userid/emp_resume.out -q del
```

In this example:

- Only non-LOB columns from emp_resume table are selected (`select empno,resume_format from emp_resume`)
- emp_resume.del file contains the query output in delimited ASCII format (**-q del** option), **,** is the default column delimiter and **|** is the default char delimiter
- emp_resume.out contains the query statistics

2.3.4 DB2 Import utility

The Import utility inserts data from an input file or a named pipe into a table or updatable view. The Import utility uses the SQL INSERT statement to write data from an input file into a specific table or view. If the target table or view already contains data, you can either replace or append to the existing data.

The following authorization is needed when using the Import utility to:

- Create a new table you must at least have CREATETAB for the database
- Replace data you must have SYSADM, DBADM or CONTROL
- Append data you must have SELECT and INSERT

The Import utility can be invoked through:

- The command line processor (CLP)
- The Import notebook in the Control Center
- An application programming interface (API)

The following information is required when importing data:

- The path and the name of the source file
- The name or alias of the target table or view
- The format of the data (IXF, DEL, ASC or WSF) in the source file
- Mode:
  - Insert
  - Replace
  - Update, if primary key matches are found
  - create

Among other options you can also specify:

- Commit frequency
- Number of records to skip from input file before starting to Import

The following is an example of the IMPORT command issued through the CLP window:

```
import from stafftab.ixf of ixf insert messages impstaffmsgs.txt into userid.staff
```

Attention: When creating a table from an IXF file, not all attributes of the original table are preserved. For example, referential constraints, foreign key definitions, and user-defined data types are not retained.

See Chapter 5., “Export and Import with DB2 distributed” on page 79 for restrictions that applies to the Import utility. For more details, see Chapter 2 of *IBM DB2 UDB Data Movement Utilities Guide and Reference*, SC09-4830.

The import utility can be used to insert data from a file or a named pipe to a table in a

- Distributed database
- Mainframe database through DB2 Connect (only IXF format)

Note: For performance, use the Load utility on distributed wherever it is possible, except for small amounts of data.

2.3.5 DB2 Load utility

The Load utility loads data into an existing DB2 table from a file residing on the same server as the database, or on a remotely connected client. The data can also be loaded from a user

defined cursor. This capability, a new Load option with DB2 V8, is often referred to as Cross Loader.

The following information is required when loading data:

- The name of the source file
- The name of the target table
- The format (DEL, ASC, IXF or CURSOR) of the source file
- If the input data is to be appended or to replace the contents of the table

The following is an example of Load from a file. The command specifies a message file, that tempfiles are to be used, the path, replace option, and the table to be loaded:

```
load from stafftab.ixf of ixf messages loastaff.msgs
tempfiles path /u/myuser replace into staff
```

Important: The Load utility does not fire triggers, and does not perform referential or table constraints checking. It does validate the uniqueness of the indexes.

The following authorization is needed when using the Load utility:

- SYSADM, DBADM or Load on the database
- Privileges needed depending on the Load mode:
  - insert: INSERT on the table
  - replace: INSERT and DELETE on the table

The Load utility can be invoked through:

- The command line processor (CLP)
- The Load notebook in the Control Centre
- An application programming interface (API)

See Chapter 6, “Load with DB2 Distributed” on page 91 for restrictions that apply to the Load utility. For more details, see Chapter 3 of *IBM DB2 UDB Data Movement Utilities Guide and Reference,* SC09-4830.

The Load utility loads data from a file or a named pipe into a table a:

- Local distributed database where the load runs
- Remote distributed database through a locally cataloged version using the CLIENT option

The Load utility is faster than the Import utility, because it writes formatted pages directly into the database, while the Import utility performs SQL INSERTs.

Attention: Data Propagator does not capture changes in data done through the Load utility.

2.3.6 DB2 Cross Loader option

The Cross Loader is another name for the new cursor load capability in the Load utility in DB2 UDB V8. This option allows you to transfer data from one location for loading it into a DB2 table in the same or other location within a single utility job execution.

By specifying the CURSOR file type when using the Load utility, you can load the results of an SQL query directly into a target table without creating an intermediate exported file.

By referencing a nickname within the SQL query, the Load utility can also load data from another database in a single step. Examples are listed in Appendix A, “Defining a Federated Database” on page 299.

To execute a load from cursor operation from the CLP, a cursor must first be declared against an SQL query. Once this is done, you can issue the LOAD command using the declared cursor’s name as the cursorname and CURSOR as the file type. The following CLP commands will load all the data from your.TABLE1 into my.TABLE1:

```
DECLARE mycurs CURSOR FOR SELECT * FROM your.table1
LOAD FROM mycurs OF cursor INSERT INTO my.table1
```

The Cross Loader function can be invoked through:

- The command line processor (CLP)
- An application programming interface (API)
- The Control Center

See Chapter 5., “Export and Import with DB2 distributed” on page 79 for restrictions that apply to the Import utility. For more details, see Chapter 3 of the *IBM DB2 UDB Data Movement Utilities Guide and Reference,* SC09-4830.

The Cross Loader can read data from a table residing in a:

- Distributed database
- Nickname representing a connected source table

The target table has to be in the same database as the source table or the source nickname.

This is a very flexible way of moving data within the distributed platforms.

2.3.7 DB2 db2move

This command facilitates the movement of large numbers of tables between DB2 databases located on the distributed platforms.

The tool queries the system catalog tables for a particular database and compiles a list of all user tables. It then exports these tables in IXF format. The IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another platform and imported or loaded into a DB2 database on that platform.

This tool calls the DB2 Export, Import, and Load APIs, depending on the action requested by the user. Therefore, the requesting user ID must have the correct authorization required by those APIs, or the request will fail.

This tool exports, imports, or loads user-created tables. If a database is to be duplicated from one operating system to another operating system, **db2move** facilitates the movement of the tables. It is also necessary to move all other objects associated with the tables, such as: aliases, views, triggers, user-defined functions, and so on.

The load action must be run locally on the machine where the database and the data file reside. A full database backup, or a table space backup, is required to take the table space out of backup pending state.

For details about the Export, Import and the Load utility, see:

- Chapter 5, “Export and Import with DB2 distributed” on page 79
- Chapter 6, “Load with DB2 Distributed” on page 91

For details about the **db2move** command; see *IBM DB2 Universal Database Command Reference*, SC09-2951-01.

DB2 UDB **db2move** is a common command and option interface to invoke the three utilities mentioned above.

2.3.8 DB2 db2look

Based on the information in the DB2 system catalog, **db2look** generates the required DDL statements to reproduce the database objects of a distributed database. It can also generate DDL customized for distributed databases for some of the objects (tables, views, and indexes) in a mainframe database.

db2look can also generate update statement for the catalog statistics and the configuration parameters for the system.

This tool can also generate the required UPDATE statements to replicate the statistics on the objects in a test database, as well as statements for the update of the database configuration and database manager configuration parameters.

We show two simple examples:

- The first one is a command that generates the DDL statements for objects created by user *walid* in database *DEPARTMENT*. The **db2look** output is sent to file db2look.sql:

  ```
  db2look -d department -u walid -e -o db2look.sql
  ```

- The second one is a command that generates the UPDATE statements for the database and database manager configuration parameters, as well as the db2set statements for the registry variables in database *DEPARTMENT*. The **db2look** output is sent to file db2look.sql:

  ```
  db2look -d department -f -o db2look.sql
  ```

To execute these commands you need SELECT authorization on the system catalogs.

The db2look function can be invoked through:

- The command line processor (CLP)
- The Control Center

For details, see Chapter 9, “Getting ready for moving data” on page 213, and *IBM DB2 Universal Database Command Reference*, SC09-2951-01.

db2look is not a tool for moving data, but we highly recommend it for moving complete or partial *data definitions* when this is needed.

2.4 Tools for z/OS and multiplatforms

In this section we provide a brief overview of IBM tools that can be used to move DB2 data.

2.4.1 DB2 Administration Tool for z/OS

IBM DB2 Administration Tool for z/OS (DB2 Admin for short) offers a comprehensive set of database administration functions that helps in efficiently managing DB2 UDB for z/OS and OS/390 subsystems. Designed with intuitive, interactive functionality, DB2 Admin relieves the DBAs from time consuming tasks.

DB2 Admin helps you with:

- **Security management**

  You can display authorizations that have been granted on any type of DB2 object, and REVOKE these authorizations or GRANT new ones. The tool also provides REVOKE

impact analysis to thwart inadvertent data loss when dropping tables. You can display the list of secondary auth IDs, as well as to change your SQLID.

- **Performance management**

  DB2 Admin has a built-in EXPLAIN function, which allows you to EXPLAIN a query and provides an interpretation of the PLAN_TABLE output into English sentences. A set of performance health check catalog queries is also provided.

- **System management**

  DB2 Admin provides in-depth DB2 catalog navigation: objects in the catalog are displayed and interpreted, and relevant catalog information is presented logically.

  DB2 Admin allows you to display and cancel threads; display and alter buffer pools; and display, start and stop DB2 traces. It also provides a very convenient way to administer RLF and DDF tables. You can also display the current DSNZPARMs, change parameters, generate new DSNZPARM modules with changes, and activate those changes in DB2.

  DB2 Admin is integrated with DB2 utilities to simplify the creation of DB2 utility jobs. This includes TEMPLATE and LISTDEF support, even for customers that are not yet on DB2 Version 7, where DB2 Admin emulates these functions.

- **Application management**

  Admin allows you to work with a copy of the actual DB2 catalog to avoid contention on the real DB2 catalog. You can also access a remote DB2 catalog, via a DDF connection, enabling centralized management of all your DB2 subsystem through a single DB2 Admin session.

  From DB2 Admin you can execute dynamic SQL statements, or invoke SPUFI.

  Its integration with other DB2 tools creates additional functionality with product-specific line commands for table editing, SQL cost analysis, and path check analysis. Through its DB2 tools launch pad, DB2 Admin is the central access point for any tool with an ISPF interface.

  You can issue any DB2 command including BIND, REBIND, and FREE selected plans and packages.

The DB2 Admin ALTER and MIGRATE functions can simplify administration tasks. After using the ALTER function to specify desired changes, the tool generates the jobs required to implement these changes. These jobs unload the data, recreate the table, and reload the data. The tool handles all object dependencies during ALTER and MIGRATE. And, after the MIGRATE function has defined the target DB2 subsystem, DB2 Admin creates the jobs needed to copy definitions and data to the target:

- The ALTER function lets you change the name and attributes of a table or column, insert new columns, and drop existing columns. The ALTER of primary key characteristics can be propagated to foreign keys.
- The MIGRATE function facilitates the copying of all the objects and data in one or more databases or table spaces to another DB2 subsystem.

Prompt options can be activated for five types of statements: definition, authorization, and update SQL, DB2 commands, and DSN commands. These options enable you to edit or execute the statements, put them in work statements or run them in batch jobs.

Using DB2 Admin to move data

If you are moving DB2 data to another subsystem, you have to make sure that you can access the DDL files, as well as unload files from the other subsystem, so that you are able to use them in the new subsystem.

You can use the DROP and REVOKE impact analysis, and the disk space estimation provided by DB2 Admin for the preparation and understanding of the needs for the objects to be recreated at the target system.

When dealing with the need to move DB2 data, the MIGRATE function of the DB2 Admin tool facilitates the copying of all the object definitions and data in one or more databases or table spaces to the same or another DB2 subsystem. Admin's cloning capability allows one to extract the DDL (and data) from a source DB2 system, move the definitions to a target system, and change the DB2 identifiers (such as owner, name, or dbname) tailoring them to the naming standards of the target system.

For instance, you can use MIG to copy a whole DB2 database. Migrate is done in three steps:

1. Fill in the migrate panels to generate jobs.
2. Run jobs on source system.
3. Run jobs on target system.

For the general use of panels and for migration examples, see *DB2 for z/OS DM Tools for Database Administration and Change Management*, SG24-6420-00.

When using the MIGrate command, you can use the **ADD** command to add new database, or tables from another database, to be migrated at the same time. You can change the database name, as well as the owner and storage group for indexes and table spaces. You can choose whether you want to migrate only the DDL, the data, or both.

If you are dealing with several databases and several objects, make sure to have PTF UQ72062 for APAR PQ68028 applied. It provides enhancements to MIGRATE by allowing the creation of work statement lists, the cloning of these statements, and the masking of names.

MIGRATE generates jobs to perform the requested tasks. It can drop the existing target database at your request, as image copies and run check data and runstats. When the target database is in the same subsystem as the source database, you can submit the jobs locally.

For more details on using MIGRATE locally or remotely, see *DB2 Administration Tool for z/OS User's Guide*, SC27-1601-02.

We have used this tool for the task of generating DDL, see Chapter 9, "Getting ready for moving data" on page 213.

2.4.2 DB2 Data Export Facility tool for z/OS

Data Export Facility tool captures a subset of your DB2 production data on z/OS to be moved to your development and test environments.

DB2 Data Export Facility provides an interface for extracting data based either on referential structures defined in DB2 or on a simple table.

When operating on referential structures you can start anywhere in a referential set and navigate anywhere throughout the set. The tool allows easy selection of the desired scope of data among the referential set. This can be a table, a table and all its dependencies, or only certain dependencies.The result will be referentially intact sets of data.

When operating on one single table, the tool can:

- Select only certain columns and rows
- Reorder columns according to the target table
- Select only partial set of rows
- Transform data values

- Masking sensitive data

When you tell DEF that you want to export data it will build a file with SQL in it to extract the data. The extracted data is placed in a file (or files). The data is written to these files in a format that could be used as input to the DB2 Load utility. It also optionally builds the LOAD control card for each table in the RI set. It generates a job to do all of this.

Note: DB2 Data Export Facility does not create the DDL for the target subsystem. You must use another product such as DB2 Automation tool to prepare the target environment to receive the data exported by DB2 Data Export Facility.

For details, see *DB2 Data Export Facility for z/OS User's Guide*, SC27-1466-00.

2.4.3 DB2 High Performance Unload tool for z/OS

IBM DB2 High Performance Unload (HPU) works primarily outside DB2 when unloading DB2 tables from either a table space or an image copy.

The basic part of HPU is an UNLOAD command and an optional SELECT statement, compatible with the syntax of the DB2 SELECT statement.

Partitioned table spaces can be unloaded in parallel by partition to multiple output files.

HPU can:

- Do multiple unloads of the same table space or image copy
- Help you to manage and control the unload activity
- Work outside DB2 and access the:
  - VSAM files that contain the table space
  - Sequential files that contains the image copy data set

HPU can use the following sources for input data:

- Partitioned and non partitioned table spaces
- Full or incremental image copies

HPU scans a table space and creates the output file in the format you specify. The output format can be:

- DSNTIAUL compatible
- VARIABLE, variable length records
- DELIMITED, a delimited file
- USER, choice free conversion of the output

Example of unloading all rows from table myTable:

```
UNLOAD TABLESPACE mydb.myTS
   SELECT * FROM me.myTable
   OUTDDN (DDNTBL02)
   FORMAT DSNTIAUL
```

HPU can do the following:

- Parallel execution of several unloads accessing the same table space
- Unload selected rows and columns
- Unload every n rows and a maximum number of rows
- Generate Load control statements for subsequent reload
- Using the HPU user exit, you can inspect, modify, or discard DB2 rows
- Translate between EBCDIC, ASCII, UNICODE, and CCSID.

You can code as many SELECT statements as necessary for tables belonging to the same table space. Different output files can be created during the same unload process.

Note: The translation from EBCDIC to ASCII and from ASCII to EBCDIC is supported via the translation values provided in SYSIBM.SYSSTRINGS table.

HPU can also be used either in batch or interactive mode via panels from the DB2 Administration tool.

The following authorization is needed:

- At least SELECT on the actual table
- RACF read on actual image copy data set

For details, see *Chapter 7, "IBM DB2 High Performance Unload for z/OS" on page 119,* and *IBM DB2 High Performance Unload for z/OS Version 2, Release 1 User's Guide*, SC27-1602-00.

DB2 High Performance Unload tool is a valid alternative to the DB2 Unload utility:

- Performance tests shows a slightly better performance than the Unload utility and a considerable less CPU time in execution. To gain this performance advantage it is necessary to use the native HPU, that is it should be executed without using the standard SQL via the DB2 engine.
- HPU provides you with more versatility than Unload because you can use the SQL select statement to select the rows and columns that you want to be unloaded.
- HPU can unload from an incremental image copy.
- HPU also provides you with more flexibility regarding customizing the output through the USER DEFINED format.
- The DELIMTED format of HPU is compatible with the DEL format on distributed platform

For recent technical information on HPU, and also comparisons with Unload, specify `High Performance Unload` in the search field at the Web site:

`http://www.ibm.com/support/search`

2.4.4 DB2 Data Replication tools for z/OS and Multiplatform

The IBM Replication tools are a set of DB2 Data Propagator programs and DB2 UDB Replication tools that copy data between DB2 databases.

Note: Data can also be replicated from non-IBM relational database management systems by use of a Federated Database.

The tool replicates changes in data across the enterprise motivated by the need of:

- Distributing data to other systems or locations
- Consolidating data from other systems or locations
- Auditing changes to sensitive data

It can be used in the following situations:

- Between DB2s on distributed platforms
- Between DBs on hosts that support DRDA connectivity
- Between DB2s on distributed platforms and DBs on host supporting Distributed Relational Database Architecture (DRDA) connectivity

Replication allows users and applications access to production data without putting extra load on the production database. The replication tools allow you to customize the copy table structure. You can include the execution of SQL statements in the copying to the target database to customize the data being copied.

Data Propagator has a set of control tables with information about which source tables to capture changes from; information about subscribers on changes; and where to deliver the changes is also needed. Status about capturing and delivering is continuously updated.

The pre-replicating process consists of the following:

- Change the option DATA CAPTURE to YES with a ALTER DDL on the relevant source table
- Populate the control tables and create tables where to store the changes.

The Replication Center can be used to do this. The Replication Center is a graphical interface that runs on Windows and UNIX systems and it must have connectivity to both the source and the target servers.

The replicating process consists of the following sequence of events:

1. DB2 writes complete log records for all updates, inserts and deletes to the log
2. Capture program continuously:
   - Reads the logs for relevant log records for tables registered
   - Inserts before and after images of the changed data in a clone of the source table (CD-table) and extends them with log information (RBA and timestamp)
   - Updates the Capture control tables
3. Apply program activated by event or on demand:
   - Reads the changes in the CD-table
   - Inserts committed changes in a target table (CCD-table)
   - Updates the Apply control tables
4. Pruning program:
   - Deletes from the CD-table changes that have already been delivered through the apply program
   - Updates relevant control tables
5. The changed data is available for the subscriber in the target table.

For details, see *IBM DB2 Replication Guide and Reference*, SC26-9920-00, and the recent redbook *A Practical Guide to DB2 UDB Data Replication V8,* SG24-6828.

If you need to move data on a regular basis, Data Propagator is a very efficient tool whenever you are updating the target with only the changes to the source. This reduces processing and transmission. Data Propagator administers the process and the schedules of the job runs. Compared to other function and tools, Data Propagator is not suitable for a one time only data movement. It is a tool that helps you keeping your databases in sync continuously or making captured changes available. To facilitate this, a substantial preparation and customizing effort is necessary.

2.4.5 DB2 Web Query tool for z/OS and Multiplatform

DB2 Web Query tool is a Data Management tool that allows you to create and run SQL queries against your DB2 databases, through a Web browser.

DB2 Web Query tool uses Java technology to provide server platform independence, and it is running in a Java servlet application server environment.

DB2 Web Query tool allows you to run SQL queries against DB2 databases and view the results, or export them for use in other applications, such as:

- Create and run SQL queries
- Save queries and results for later use
- View query results through your web browser
- E-mail query results
- Export query results to a new or existing table
- Export query results to the following formats:
  - XML
  - HTML
  - CSV (comma separated value)
  - text files
  - Microsoft Excel
- Connect to Informix Version 9.3 databases

For details, see *IBM DB2 Web Query Tool User's Guide*, SC27-0971-05, and the recent redbook *DB2 Web Query Tool Version 1.2*, SG24-6832.

Note: The DB2 Web Query tool is product that can access DB2 data on z/OS, distributed platforms, and iSeries.

2.4.6 DB2 High Performance Unload tool for Multiplatforms

IBM High Performance Unload for Multiplatforms is a tool that works outside DB2 and unloads data. It normally runs while the DB2 database manager is running and accesses the same physical files as the DB2 database manager, but it can also unload data from a backup copy.

HPU can use the following sources for input data:

- DB2 partitioned and non partitioned table spaces
- DB2 full backup of an SMS table space

With HPU, you can:

- Unload data to flat files, tape devices or named pipes in DEL format
- Unload summary tables and ALIAS tables
- Use SELECT syntax to specify the columns and rows to unload
- Use of SELECT eliminates the need for special applications to convert data
- Limit the output by using the MAXROWS, SKIP and INTERVAL keywords
- Provides parallel processing

You can unload table data to multiple targets.

You can unload larger volumes of data than the operating system file system limit through unloading into several smaller output files.

HPU creates messages that contain detailed summaries of unload operations, including unload process statistics, results, and error descriptions.

The following example shows how to invoke the HPU; in this case we use HPU to unload data from table mytable to file myoutputfile:

```
db2hpu -d mybase -t mytable -o myoutputfile
```

The following examples show how to invoke HPU with the use of a control file containing the unload instructions for two situations:

```
db2hpu -f sampleN.cntl
```

- The sample1.ctl control file unloads *a table*:

```
GLOBAL CONNECT TO SAMPLE;
UNLOAD TABLESPACE SAMPLE.USERSPACE1
SELECT *FROM Administrator.EMPLOYEE;
OUTPUT("c:\gasample \sample01.out");
```

- The sample2.ctl control file unloads a table space from *a backup*:

```
GLOBAL CONNECT TO SAMPLE;
UNLOAD TABLESPACE USERSPACE1
backup "C:\TEMP \SAMPLE.0 \DB2 \NODE0000 \CATN0000 \20020715 \133912.001"
OUTPUT ("c:\gasample \sample2.out"REPLACE)FORMAT DEL;
```

For details, see *Chapter 8, "IBM DB2 High Performance Unload for Multiplatforms" on page 193,* and *IBM DB2 High Performance Unload for Multiplatforms User's Guide*, SC27-1623-01.

When HPU does not access the data via the DB2 engine, it is significantly faster than the DB2 EXPORT utility.

However, the use of the DB2 Export utility is recommended in the following cases:

- When you are doing selections on rows (WHERE conditions.)
- When you need to run a SELECT statement that HPU does not support, which includes SELECT statements that contain joins of multiple DB2 tables, recursive queries, or views.
- When the SELECT statement you are running could use an index (HPU does not use all indexes to access data in the table that you want to unload.)
- When you need to unload a table in a single-partition environment that is not physically located on the machine where HPU is running.

2.4.7 DB2 UDB Warehouse Manager for UNIX and Windows

The Data Warehouse Center (DWC) can be used to move data from operational databases to a warehouse database.

You can use the DWC to define the structure of the operational databases, called sources. You can then specify how the operational data is to be moved and transformed for the warehouse. You can model the structure of the tables in the warehouse database, called targets, or build the tables automatically as part of the process of defining the data movement operations. The DWC uses the following DB2 functions to move data:

- SQL to select data from sources and insert the data into targets.
- DB2 utilities to export data from a source and then use DB2 Load the target
- Replication to copy large quantities of data from

See the *Data Warehouse Center Administration Guide*, SC26-9993-01, and also *DB2 Warehouse Management: High Availability and Problem Determination Guide*, SG24-6544. We have not used this tool during this project.

2.5 Data movers summary

Table 2-1 lists all the products presented in this chapter and summarizes their scope of applicability.

Table 2-1 Overview of data movers

FUNCTION	Platform	Paragraph	"Unload"		"Load"		Enc. sch.	Format	SQL-stmts	Backup	Detail
			z/OS	Distr.	z/OS	Distr.					
DSN1COPY	M	2.2.1	X							X	
DSNTIAUL	M	2.2.2	X						X		
Reorg utility	M	2.2.3	X								
Unload utility	M	2.2.4	X				E,A,U			X	3
Load utility	M	2.2.5			X		E,A,U				4
Cross Loader	M	2.2.6	X	X	X		E,A,U		X		4.2
Administration tool	M	2.4.1	X								
Data Export facility	M	2.4.2	X						X		
HPU	M	2.4.3	X				E,A,U ,S,C	D,V,U ,T	X	X	7
Replication tools	MD	2.4.4	X	X	X	X			X		
Web Query tool	MD	2.4.5	X	X					X		
BACKUP & RESTORE	D	2.3.1		X		X				X	
Export utility	D	2.3.2	X	X				D,I	X		5.1 5.2
IMPORT UTILITY	D	2.3.4			X	X		A,D,I			5.3 5.4
LOAD UTILITY	D	2.3.5				X		A,D,I			6
CROSS LOADER	D	2.3.6	X	X		X			X		6.3.4
HPU	D	2.4.3		X			A,S,C	D	X	X	8
db2move command	D	2.3.7		X		X		I			
db2look command	D	2.3.8									
Warehouse Manager	D	2.4.7	X	X		X			X		

Platform — M = mainframe / D = distributed / MD = both

Paragraph — Refers to the sub-chapters in chapter 3

Enc. Sch. — E = EBCDIC / A=ASCII / U=UNICODE / C=CCSID / S=ASIS

Format — A = ASC (positional) / D=DEL (delimited) / I=PC/IXF / V=VARIABLE / U=USER / T=DSNTIAUL

SQL stmt — Tells if you can use SQL select to express what to be unloaded

Backup — Tells if you can unload from a image copy or a backup

Detail	Refers to chapter or sub-chapter where the function is discussed in more details than Chapter 3

Part 2

Product functions and utilities

In this part we introduce the main functions offered by the members of the DB2 Family in the area of data movement.

This part is structured in the following chapters:

- Unload with DB2 for z/OS
- Load with DB2 for z/OS
- Export and Import with DB2 distributed
- Load with DB2 Distributed

Load and Unload utilities for DB2 for z/OS and OS/390 Version 7 are made available through the DB2 Version 7 Utilities Suite.

3

Unload with DB2 for z/OS

In this chapter, we discuss the Unload utility made available with the DB2 for z/OS Utilities Suite Version 7.

We show you:

- A brief overview of the Unload utility
- The advantages it has over its predecessors
- Syntax and rules in using the Unload
- The possible options when using Unload
- Some examples on using the Unload utility

3.1 Overview of the Unload utility

The Unload utility is an alternative and a replacement offered with DB2 for z/OS V7 to the REORG UNLOAD EXTERNAL function. With Unload, you can unload rows from an entire table space or select specific partitions or tables to unload. You can also select columns by using the field specification list. If a table space is partitioned, you can unload all of the selected partitions into a single data set, or you can unload each partition in parallel into physically distinct data sets.

Unload can be used to unload data from one or more source objects to one or more BSAM sequential data sets in external formats. The source can be DB2 table spaces, DB2 Image Copy data sets.

When comparing to previous methods, the Unload utility provides:

- Enhanced functionality
- Better performance
- Improved concurrency
- Higher availability
- Low cost of operation

The online functions of Unload allow the unload of data from DB2 tables with SHRLEVEL CHANGE, which enhances the continuous availability of DB2 subsystems.

The Unload utility requires SELECT authority on the table or tables in the table space. This is similar to the DSNTIAUL unload programs, but differs from REORG UNLOAD EXTERNAL, which requires REORG utility authority.

The DISCARD option of REORG UNLOAD EXTERNAL, which can be used to discard data with the WHEN clause, is *not* supported in the Unload utility.

With the current versions of DB2, you cannot load a delimited input file into DB2. A delimited file is a sequential file that contains row and column delimiters often used to move data across the distributed platforms. With the recently announced DB2 for z/OS Version 8, you will be able to use the Load utility to load into DB2 a delimited input file from another relational database. Even more important, you will not need to write a program that converts the data into the correct format, or use INSERT processing and give up the performance advantages of the Load utility. See the following Web site for more information on DB2 for z/OS Version 8:

http://www.ibm.com/software/data/db2/os390/db2zosv8.html

3.1.1 Extra functionality of the Unload utility

Figure 3-1 summarizes all the functions of the Unload utility. On the left of the diagram you see listed the REORG UNLOAD EXTERNAL functions which Unload includes as a subset. In addition to the options provided by REORG UNLOAD EXTERNAL, the UNLOAD statement provides:

- Unload from image copy data sets created by Copy utility, both full and incremental
- Unload from inline COPY data sets created by Reorg and Load utility
- Unload from data sets created by stand-alone DSN1COPY utility
- Encoding scheme, ASCII and UNICODE
- SHRLEVEL REFERENCE and SHRLEVEL CHANGE
- Sampling, limiting of rows
- Partition parallelism

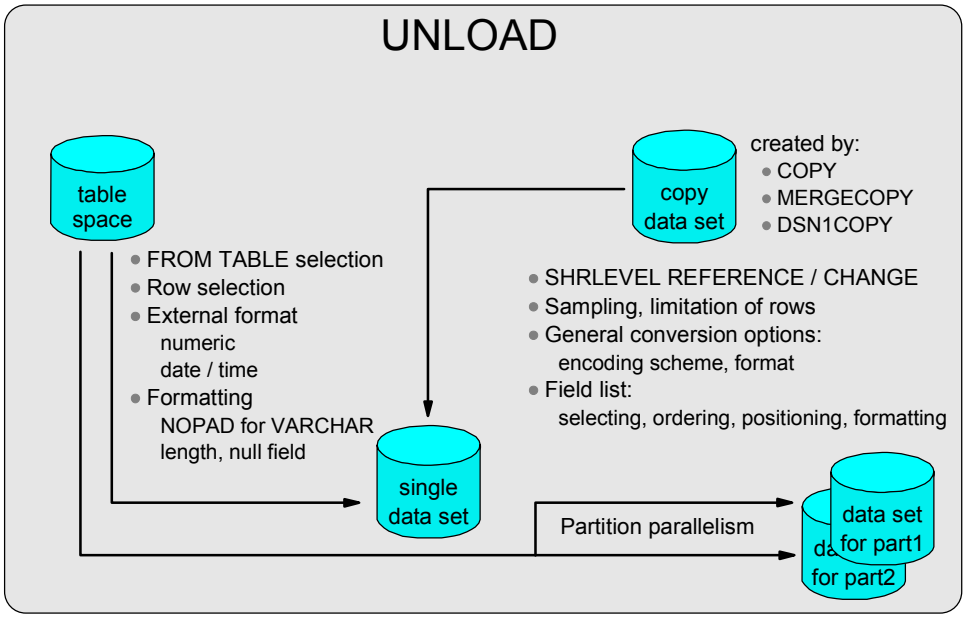

Figure 3-1 Enhanced functions

3.1.2 Privilege and authority required

To execute the Unload utility, the user needs any one of the following:

- Ownership of the tables
- SELECT privilege on the tables
- DBADM authority on the databases
- SYSADM authority
- SYSCNTL authority for catalog tables only

3.1.3 Phases of the Unload utility

There are three phases of the Unload operation.

- UTILINIT

  In this phase the Unload utility is initialized and set-up.

- UNLOAD

  In this phase, records are read from the DB2 table. Unloading records to sequential data sets. If UNLOAD is processing a table or partition, DB2 takes internal commits to provide commit points at which to restart in case of operation should halt in this phase

- UTILTERM

  In this phase, clean-up of temporary files is made

Figure 3-2 shows a diagram of the phases of the Unload operation.

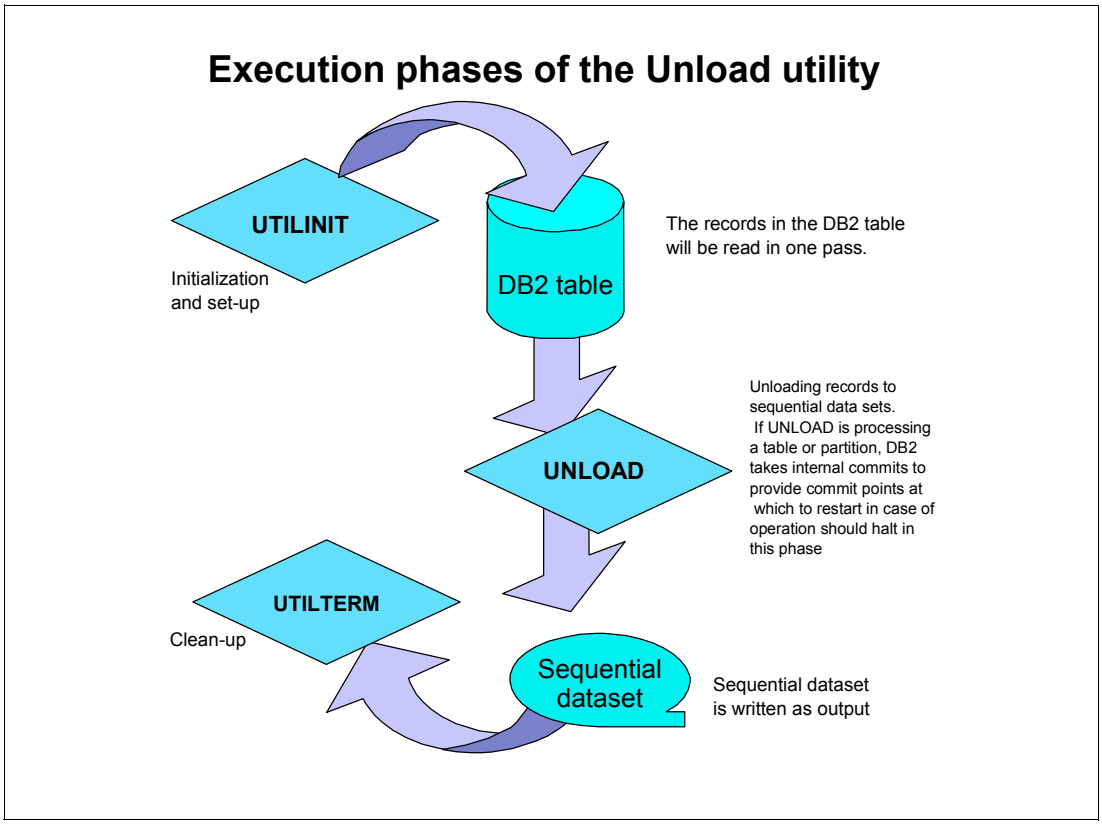

Figure 3-2 The execution phase of the Unload utility

3.2 Input and output data sets

The Unload utility can accept in input directly the table space, or the output of the different flavors of a Copy utility execution on the table space. It produces in output two data sets: one for the data, with the option of several formats, and one for the data definition. In this section we describe Unload input and output data sets.

3.2.1 Output data sets from Unload

Unload outputs two data sets, which can be used for reload with the Load utility:

- The SYSREC data set is associated with the UNLDDN option of UNLOAD. The data set contains the output data from the Unload utility, which is input to a subsequent Load utility for loading data into another table space. This data set can be defined in the JCL, and the DDNAME is provided on the UNLDDN option. A TEMPLATE command can also be used to define the data set characteristics and the template name is input in UNLDDN. The data set is dynamically created by the Unload utility with the TEMPLATE option. The TEMPLATE command is required when unloading multiple table spaces, either with multiple UNLOAD statements, or by using the LISTDEF command to supply a list of table spaces to a single UNLOAD utility statement.
- The SYSPUNCH data set is associated to the PUNCHDDN option of UNLOAD. The data set is used by the Unload utility to store the table definition (see Example 3-2 on page 39), which is used by the Load utility to load data into another table. When the punch data set is required, its allocation is similar to SYSREC either via DDNAME in the JCL or via TEMPLATE.

In this chapter we discuss each function in detail and show examples of JCL and job outputs.

3.2.2 Input of Unload from image copy

The Unload utility can be used to unload from data sets that are created by the following utilities:

- COPY
- LOAD inline image copy
- MERGECOPY
- REORG TABLESPACE inline image copy
- DSN1COPY

Image copy created by concurrent copy is not supported by Unload. Unloading data from image copy is not supported by REORG UNLOAD EXTERNAL utility and the DSNTIAUL program.

Note: The table space of the image copy *must* exist in the host DB2 subsystem for unloading from copy data sets. Copies of dropped table spaces are not supported.

The syntax for the Unload utility from image copy is shown in Figure 3-3. Here are some additional considerations when using the FROMCOPY option:

- Only a single copy data set name can be specified using the FROMCOPY option. Use FROMCOPYDDN if multiple image copies are concatenated to a DD statement.
- The table space must exist when Unload is run.
- The table space should not have been dropped and recreated since the image copy was taken (unless the OBID has been kept identical).
- If the table was altered with ALTER ADD COLUMN after the image copy was taken, Unload will set system or user defined defaults for the added column when the data is unloaded from such an image copy.
- Image copy created by Inline COPY operation (LOAD or REORG TABLESPACE) can contain duplicate pages. If duplicate pages exist, the Unload utility issues a warning message, and all the qualified rows in the duplicate pages will be unloaded into the output data set.
- If incremental copy is specified as the source, only the rows in the data set are unloaded.

FROMCOPYDDN can be used in place of FROMCOPY when multiple copies need to be specified to Unload. The copies may consist of:

- Full image copy and incremental copies. Consider using the MERGECOPY utility to merge the image copies to a single copy and input to Unload.
- Image copy created with the DSNUM option, which can be a copy of a partition or a piece of a non-partitioned table space. These pieces can be concatenated under a DDNAME to form a single input data set image.
- The order of data sets in DDNAME concatenation is important. Unload might output unpredictable results if the most recent image copy data sets and older image copies are intermixed.

Note: TEMPLATE can be used for the SYSPUNCH and SYSREC data set definitions identified as PUNCHDDN and UNLDDN options in the UNLOAD syntax respectively. LISTDEF cannot be used to pass a list to the LIST option to specify image copy data sets.

3.3 Unload syntax and examples

The Unload statement allows a multitude of options, Figure 3-3 shows just the main part of the syntax diagram. For a list of each parameter you can refer to Appendix D, “DB2 UDB for z/OS Unload options” on page 323, or for more details see *DB2 UDB for OS/390 and z/OS Version 7 Utility Guide and Reference,* SC26-9945-03.

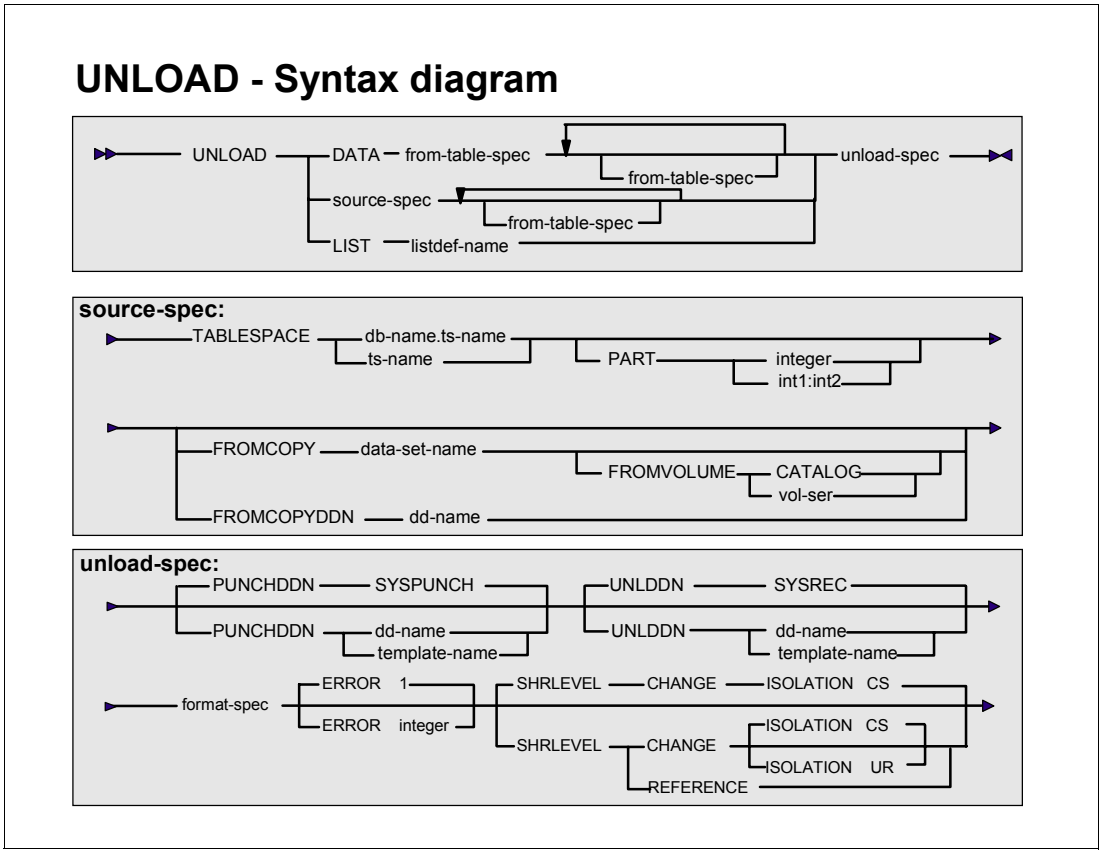

Figure 3-3 UNLOAD — Syntax diagram, main part

3.3.1 Examples of using the Unload utility

The examples shown in this section illustrate the new function of the Unload utility.

Unload from an image copy

This an example of an Unload from an image copy of CUSTOMER data. The image copy data set was created by the COPY utility with SHRLEVEL CHANGE. The job produces two data sets associated to SYSPUNCH and SYSREC respectively. The FROMCOPY points to an image copy data set. See Example 3-1.

Example 3-1 Unload from an image copy data set

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD
//SYSIN    DD  *
  TEMPLATE ULDDDN
     DSN(DB2V710G.&DB..&TS..UNLOAD)
     UNIT(SYSDA) SPACE(45,45) CYL DISP(NEW,CATLG,CATLG)
     VOLUMES(SBOX57,SBOX60)
  TEMPLATE PNHDDN
     DSN(DB2V710G.&DB..&TS..PUNCH)
```

```
     UNIT(SYSDA) SPACE(45,45) CYL DISP(NEW,CATLG,CATLG)
     VOLUMES(SBOX57,SBOX60)
  UNLOAD TABLESPACE U7G01T11.TSCUST
         FROMCOPY DB2V710G.U7G01T11.TSCUST.D2001145.T022853L
         PUNCHDDN PNHDDN  UNLDDN ULDDDN
```

The contents of the data set associated with PUNCHDDN (SYSPUNCH) is displayed in Example 3-2. If the space allocation is not specified in the TEMPLATE statement (refer back to Example 3-1), then DB2 calculates the space requirements using the formulas:

```
SYSPUNCH = ((#tables * 10) + (#cols * 2)) * 80 bytes
SYSREC = ((high used RBA) + (#records * (12 + length of longest clustering key)) bytes
```

Example 3-2 Contents of SYSPUNCH data set

```
TEMPLATE U4851714
     DSN(DB2V710G.&DB..&TS..UNLOAD1)
     DISP(OLD,CATLG,CATLG)
LOAD DATA INDDN U4851714 LOG NO  RESUME YES
 EBCDIC  CCSID(01140,00000,00000)
 INTO TABLE "PAOLOR1 "."CUSTOMER          "
 WHEN(00001:00002 = X'0011')
 ( "C_CUSTKEY         " POSITION(  00003:00006) INTEGER
 , "C_NAME            " POSITION(  00007:00033) VARCHAR
 , "C_ADDRESS         " POSITION(  00034:00075) VARCHAR
 , "C_NATIONKEY       " POSITION(  00076:00079) INTEGER
 , "C_PHONE           " POSITION(  00080:00094) CHAR(015)
 , "C_ACCTBAL         " POSITION(  00095:00098) FLOAT(21)
 , "C_MKTSEGMENT      " POSITION(  00099:00108) CHAR(010)
 , "C_COMMENT         " POSITION(  00109:00227) VARCHAR
 )
```

UNLOAD using FROMCOPY and FROM TABLE

The FROMCOPY option unloads all tables belonging to a table space. In cases where a table space contains more than one table, and the requirement is to unload only a single table, then the FROM TABLE can be used to unload only the selected table. Example 3-3 shows the syntax for unloading from image copy for only a single table. The example also highlights the options to unload only three columns out of the eight columns defined on the table (Example 3-2) with the WHEN clause, which can be used to reduce the number of rows to unload.

A further filter on the volume of unloaded data can be achieved with the SAMPLE option. SAMPLE indicates the percentage of data to be unloaded. If the WHEN clause is used to unload selective rows, then SAMPLE is applied only on rows qualified by the WHEN selection condition.

Note: The sampling is applied per individual table. If the rows from multiple tables are unloaded with sampling enabled, the referential integrity between the tables might be lost

Example 3-3 Unload using FROMCOPY and FROM TABLE options

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD
//SYSIN    DD  *
  TEMPLATE ULDDDN
     DSN(DB2V710G.&DB..&TS..UNLOAD)
     UNIT(SYSDA) SPACE(45,45) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
```

```
TEMPLATE PNHDDN
   DSN(DB2V710G.&DB..&TS..PUNCH)
   UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
   VOLUMES(SBOX57,SBOX60)
UNLOAD TABLESPACE U7G01T11.TSCUST
       FROMCOPY DB2V710G.U7G01T11.TSCUST.D2001145.T022853L
       PUNCHDDN PNHDDN  UNLDDN ULDDDN
       FROM TABLE PAOLOR1.CUSTOMER SAMPLE 50.0
            (C_CUSTKEY,C_NAME,C_ADDRESS)
            WHEN ( C_CUSTKEY > 1000 )
```

Image copy from compressed table space

Compressed rows from image copy data set can be unloaded only when the dictionary for decompression has been retrieved. Unload ignores a compressed row with a warning message if the dictionary pages have not been read when the compressed row is encountered. An error counter is incremented, and Unload terminates with an error message if the error count exceeds MAXERR.

If the image copy is an incremental copy, or a copy of a partition or partitions, then the compression dictionary pages must exist in the same data set, otherwise Unload will fail and DB2 will issue an error message.

Advantages of Unload using image copy

Figure 3-4 summarizes the advantages of unloading from image copies:

- Unloading from image copy does not interfere with the host table space and table. No locks are taken on table space and index spaces, thus avoiding lock contentions. The only reference is to the DB2 catalog for table definitions.
- The status of the table space does not affect the Unload utility when unloaded from an image copy. The table space may be in STOP status or other restrict status.
- Either all columns of a table or a subset of columns of table can be unloaded using the FROM TABLE option. Data selection can be further qualified by the WHEN option and the the SAMPLE option.

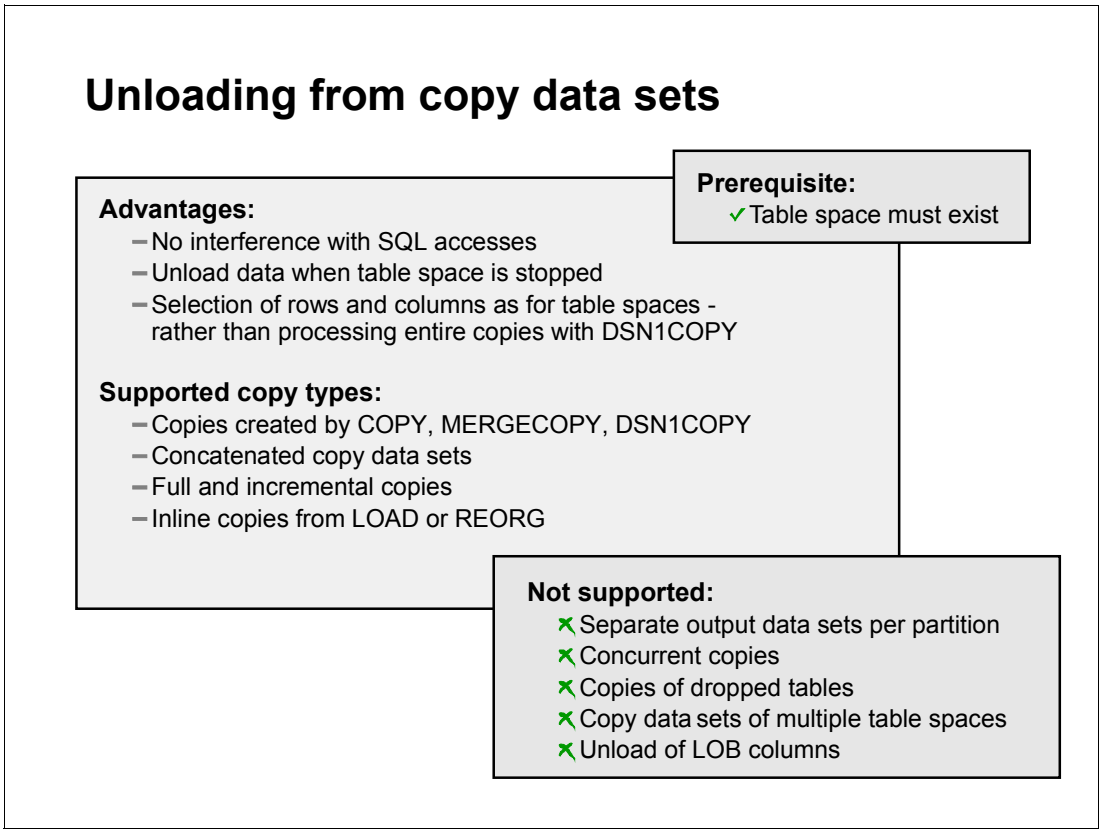

Figure 3-4 Summary of Unloading from copy data sets

Unload data from table space

In this section we discuss the Unload utility to unload data directly from a table space with SHRLEVEL CHANGE and SHRLEVEL REFERENCE. The SHRLEVEL option improves concurrency when compared to REORG UNLOAD EXTERNAL. Figure 3-5 summarizes all the functions of Unload from tables space. We explain each function in detail with examples.

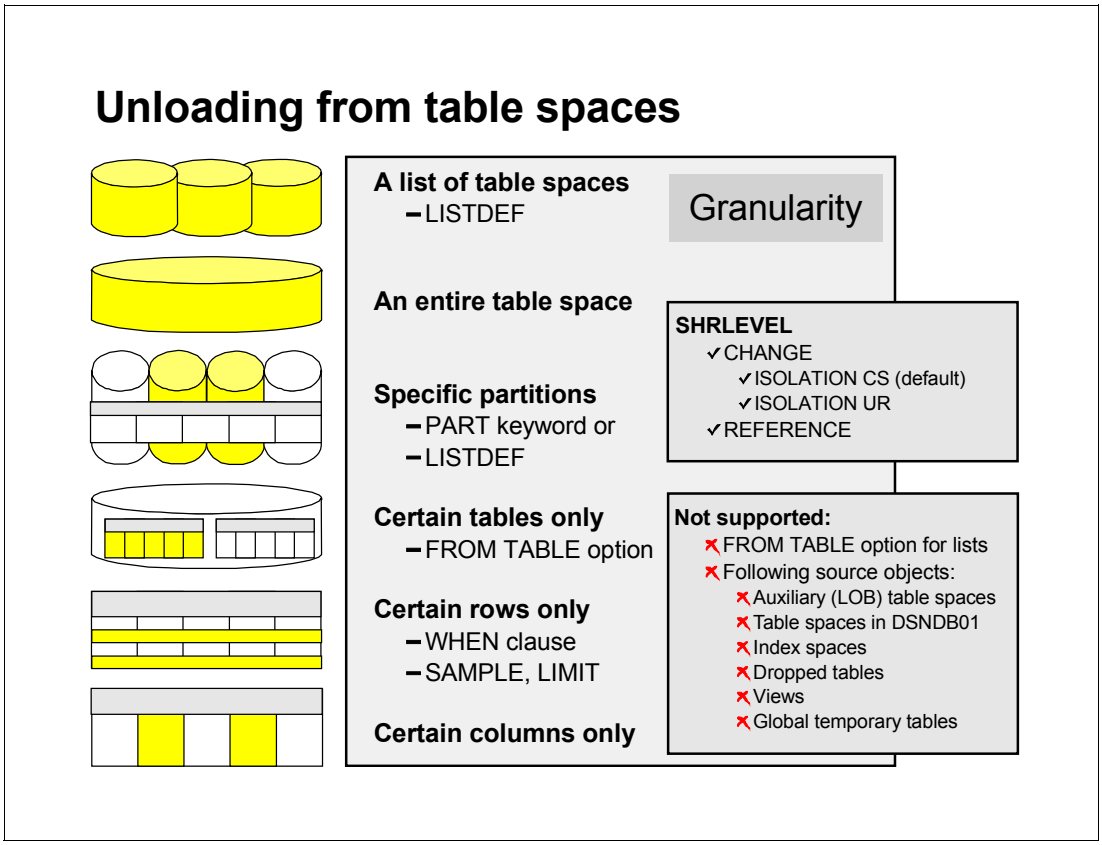

Figure 3-5 Unload from table space

Unloading from a list of table spaces

Unloading a list of table spaces and the related tables can be done using the LISTDEF command as in Example 3-4. Here we are unloading all tables in table spaces beginning with TS* in database U7G01T11. The LISTDEF will expand the wildcard U7G01T11.TS* to the appropriate table space names and group them under the label UNLIST. This list is then passed to Unload utility as UNLOAD LIST list-name. All the SYSPUNCH and SYSREC data sets are dynamically allocated using the TEMPLATE command. At the end of the Unload utility, two data sets of the form DB2V710G.RAMA.TS*.PUNCH and DB2V710G.RAMA.TS*.UNLOAD are created for each table space that is processed by the Unload utility. A sample content of the SYSPUNCH data set is displayed in Example 3-2 on page 39.

When unloading multiple table spaces with a LISTDEF, you must also define a data set TEMPLATE that corresponds to all the table spaces and specify the *template-name* in the UNLDDN option.

Restrictions

Index spaces, LOB table spaces and directory objects must not be included in the LISTDEF definition. The FROM TABLE-spec of Unload is not supported with the LIST option.

Note: Unloading from a list of table spaces is fully supported by REORG UNLOAD EXTERNAL.

Example 3-4 Unload list of table spaces with LISTDEF

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD,
//         UTSTATS=''
//SYSIN    DD  *
  TEMPLATE ULDDDN
     DSN(DB2V710G.RAMA.&TS..UNLOAD)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  TEMPLATE PNHDDN
     DSN(DB2V710G.RAMA.&TS..PUNCH)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  LISTDEF UNLIST
  INCLUDE TABLESPACE U7G01T11.TS*
  UNLOAD LIST UNLIST
         PUNCHDDN PNHDDN  UNLDDN ULDDDN
```

Unload by partition and parallelism

Data can be unloaded by partition by specifying the PART option in the UNLOAD statement. When using the LISTDEF command, specify PARTLEVEL. An output data set must be allocated for each partition for the UNLOAD to use parallel tasks to unload the data by partition. The number of parallel tasks is limited by the number of CPUs in the LPAR and the number of partitions.

UNLOAD does not activate parallel unloads if only a single output data set is allocated to each table space even though the PART or PARTLEVEL option is coded in the UNLOAD utility statement or LISTDEF command respectively. TEMPLATE can be used to dynamically allocate an output data set per partition by using the &PA key word.

Example 3-5 Sample Unload job for partition table space and parallelism

```
TEMPLATE ULDDDN
     DSN(DB2V710G.RAMA.&TS..P&PA..UNLOAD)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  TEMPLATE PNHDDN
     DSN(DB2V710G.RAMA.&TS..PUNCH)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  LISTDEF UNLIST
  INCLUDE TABLESPACE U7G01T11.TS* PARTLEVEL
  UNLOAD LIST UNLIST
         PUNCHDDN PNHDDN UNLDDN ULDDDN
```

In Example 3-5 we unloaded a list of table spaces using the LISTDEF wildcard specification and PARTLEVEL. We also allocated the output data sets using TEMPLATE with &PA in the DSN definitions. It can be seen in the output of Example 3-6 that DB2 has used two parallel tasks to unload the data by partition. UNLOAD has also created three data sets via TEMPLATE definitions as output data sets, one for each partition. If the &PA key word was not allocated to the DSN of TEMPLATE ULDDDN, then DB2 would have allocated only a single output data set, and the unload of data from partitions would be done in sequence.

Example 3-6 Sample Unload output by partition and parallelism

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = UNLOAD
DSNU050I    DSNUGUTC -  TEMPLATE ULDDDN DSN(DB2V710G.RAMA.&TS..P&PA..UNLOAD) UNIT(SYSDA)
CYL DISP(NEW,CATLG,DELETE)
  VOLUMES(SBOX57,SBOX60)
 DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
```

```
DSNU050I    DSNUGUTC -  TEMPLATE PNHDDN DSN(DB2V710G.RAMA.&TS..PUNCH) UNIT(SYSDA) CYL
DISP(NEW,CATLG,DELETE)
 VOLUMES(SBOX57,SBOX60)
 DSNU1035I   DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  LISTDEF UNLIST INCLUDE TABLESPACE U7G01T11.TS* PARTLEVEL
DSNU1035I   DSNUILDR - LISTDEF STATEMENT PROCESSED SUCCESSFULLY
DSNU050I    DSNUGUTC -  UNLOAD LIST UNLIST PUNCHDDN PNHDDN UNLDDN ULDDDN
DSNU1039I   DSNUGULM - PROCESSING LIST ITEM: TABLESPACE U7G01T11.TSPSUPP1 PARTITION 1
DSNU1039I   DSNUGULM - PROCESSING LIST ITEM: TABLESPACE U7G01T11.TSPSUPP1 PARTITION 2
DSNU1039I   DSNUGULM - PROCESSING LIST ITEM: TABLESPACE U7G01T11.TSPSUPP1 PARTITION 3
DSNU1201I   DSNUUNLD - PARTITIONS WILL BE UNLOADED IN PARALLEL, NUMBER OF TASKS = 2
DSNU397I    DSNUUNLD - NUMBER OF TASKS CONSTRAINED BY CPUS
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=PNHDDN
                       DDNAME=SYS00003
                       DSN=DB2V710G.RAMA.TSPSUPP1.PUNCH
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=ULDDDN
                       DDNAME=SYS00004
                       DSN=DB2V710G.RAMA.TSPSUPP1.P00001.UNLOAD
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=ULDDDN
                       DDNAME=SYS00005
                       DSN=DB2V710G.RAMA.TSPSUPP1.P00002.UNLOAD
DSNU251I    DSNUULQB - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=108000 FOR
TABLESPACE U7G01T11.TSPSUPP1
PART 1
DSNU1038I   DSNUGDYN - DATASET ALLOCATED.  TEMPLATE=ULDDDN
                       DDNAME=SYS00006
                       DSN=DB2V710G.RAMA.TSPSUPP1.P00003.UNLOAD
DSNU251I    DSNUULQB - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=108000 FOR
TABLESPACE U7G01T11.TSPSUPP1
 PART 2
 DSNU251I    DSNUULQB - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=446421 FOR
TABLESPACE U7G01T11.TSPSUPP1
 PART 3
 DSNU253I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=662421 FOR
TABLE PAOLOR4.PARTSUPP1
 DSNU252I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=662421 FOR
TABLESPACE U7G01T11.TSPSUPP1
 DSNU250I    DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:18
```

UNLOAD with SHRLEVEL

One of the interesting features of Unload utility is the ability to unload data from table spaces with SHRLEVEL CHANGE or REFERENCE. This feature is not available with REORG UNLOAD EXTERNAL. Users require SELECT authority on the tables in the table space.

SHRLEVEL CHANGE

SHRLEVEL CHANGE allows users to update the tables in the table space while data is being unloaded. When data is fetched from the table space with ISOLATION CS, the Unload utility assumes CURRENTDATA(NO). This ensures that uncommitted data is not unloaded and retains data currency. Data can also be unloaded with ISOLATION UR, where any uncommitted data will be unloaded. No locks are taken on the objects; this allows other DB2 operations to continue on the objects from which the data is being unloaded.

SHRLEVEL REFERENCE

This operation allows users to read the data during the unload. All writers are drained from the table space before commencement of the unload. When data is unloaded from multiple partitions, the drain lock will be obtained for all of the selected partitions in the UTILINIT phase.

Unload from table space using the FROM TABLE option

When data is unloaded from a single table space or partitioned table space, the FROM TABLE can be used to selectively unload data for a particular table from the table space. This is particularly useful when a table space contains multiple tables and the user is interested in one or more tables only. In Example 3-7 we unload three of the fourteen tables defined in SYSDBASE table space. Only a single set of SYSPUNCH and SYSREC data sets are created by the UNLOAD. The first two bytes of each record in the output data set identifies the OBID of the table. The Load utility uses the WHEN option to identify the output data related to each table. Please refer back to Example 3-2 on page 39 for sample contents of the SYSPUNCH data set and the WHEN option of the Load utility.

Using the WHEN option

The WHEN option can be used in association with FROM TABLE to unload data from the table that satisfies the *selection-condition* that follows the WHEN option. The selection-condition specifies a condition that is true, false, or unknown about a given row. When the condition is true, the row qualifies for Unload. When the condition is false or unknown, the row does not qualify. The WHEN condition can be used to unload both FROMCOPY of an image copy and from a table space. For a complete description of the selection condition, please refer to *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-00.

In Example 3-7 we also show the usage of the WHEN option with selection criteria. Each WHEN option applies to the FROM TABLE option only. Here we unload rows from SYSTABLEPART, SYSTABLES and SYSTABLESPACE where the WHEN option explicitly qualifies the data selection criteria for each table respectively.

Using the SAMPLE option

The SAMPLE option may be used to unload a sample percentage of rows from the table specified by FROM TABLE option. When SAMPLE is used in association with WHEN option, the sampling is applied to rows that qualify the selection criteria of the WHEN option. The user may specify explicitly a sample value for each table; the default is 100%.

Using the LIMIT option

Note: If the rows from multiple tables are unloaded with sampling enabled, the referential integrity between the tables may be lost.

The LIMIT option can be used to limit the total number of records unloaded from a table. If the number of unloaded rows reaches the specified limit, message DSNU1202 is issued for the table, and no more rows are unloaded from the table. The process continues to unload qualified rows from the other tables.

When partition parallelism is activated, the LIMIT option is applied to each partition instead of the entire table.

In Example 3-7 we show an example of using SAMPLE and LIMIT options.

Note: If multiple tables are unloaded from with the LIMIT option, the referential integrity between the tables may be lost.

Example 3-7 Unload selective tables from SYSDBASE using FROM TABLE

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD,
//         UTSTATS=''
//SYSIN    DD  *
```

```
  TEMPLATE PNHDDN
     DSN(DB2V710G.RAMA.&TS..PUNCH)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  TEMPLATE ULDDDN
     DSN(DB2V710G.RAMA.&TS..UNLOAD)
     UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
     VOLUMES(SBOX57,SBOX60)
  UNLOAD TABLESPACE DSNDB06.SYSDBASE
         PUNCHDDN PNHDDN  UNLDDN ULDDDN
         SHRLEVEL CHANGE ISOLATION CS
         FROM TABLE SYSIBM.SYSTABLEPART SAMPLE 50.0
          (PARTITION,
           TSNAME,
           DBNAME,
           IXNAME VARCHAR 10 STRIP BOTH TRUNCATE) WHEN
         (PARTITION=0)
         FROM TABLE SYSIBM.SYSTABLES LIMIT 100 WHEN
         (CREATOR='SYSIBM')
         FROM TABLE SYSIBM.SYSTABLESPACE WHEN
         (DBNAME='U7G01T11')

The job outputs two data sets that contain the SYSPUNCH and SYSREC information.
DB2V710G.RAMA.SYSDBASE.PUNCH
DB2V710G.RAMA.SYSDBASE.UNLOAD
```

Unload table with field selection list

Data can be unload from a table by specifying only selective fields. The default is to select all fields in the table. In Example 3-7 we select only four fields from SYSTABLEPART and all fields from SYSTABLES and SYSTABLESPACE.

Apply column functions to output field

Column functions such as STRIP, TRUNCATE, packed, ZONED and others can be applied to the output value of data fields. In Example 3-7 on page 45, we applied the STRIP function to remove all leading and trailing blanks from the VARCHAR field. We also specified the output length of the VARCHAR field as 10 bytes, which will ensure that the output data set is a fixed length.

If the length parameter is omitted, the default is the smaller of 255 and the maximum length defined on the source table column.

For a complete list and description of column functions on data types, please refer to Chapter 29 of *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-00.

Converting the character output data to other internal code

When a table space or table is created with default character set, the SCCSID value from DSNHDECP is used as the default character representation. Example 3-8 has sample CREATE statements to create database, table space, and table with CCSID EBCDIC as the default character set. Since SCCSID in DSNHDECP (macro DSNHDECM) is 37, the table DEPT will be created with the US English character set.

Tables can also be created with the European English character set of 1140 (Euro support) by changing the CCSID EBCDIC to CCSID 1140.

Example 3-8 Sample database, table space, table, and subset of DSNHDECP

```
CREATE DATABASE DSN8D71A
    STOGROUP DSN8G710
    BUFFERPOOL BP0
    CCSID EBCDIC;
CREATE TABLESPACE DSN8S71E
    IN DSN8D71A
    USING STOGROUP DSN8G710
              PRIQTY 20
              SECQTY 20
              ERASE NO
    NUMPARTS 4
       (PART 1 USING STOGROUP DSN8G710
                        PRIQTY 12
                        SECQTY 12,
        PART 3 USING STOGROUP DSN8G710
                        PRIQTY 12
                        SECQTY 12)
    LOCKSIZE PAGE LOCKMAX SYSTEM
    BUFFERPOOL BP0
    CLOSE NO
    COMPRESS YES
    CCSID EBCDIC;
CREATE TABLE DSN8710.DEPT
                (DEPTNO   CHAR(3)        NOT NULL,
                 DEPTNAME VARCHAR(36)    NOT NULL,
                 MGRNO    CHAR(6)                ,
                 ADMRDEPT CHAR(3)        NOT NULL,
                 LOCATION CHAR(16)               ,
                 PRIMARY KEY(DEPTNO))
         IN DSN8D71A.DSN8S71D
         CCSID EBCDIC;
COMMIT;

DSNHDECP sample definition
DSNHDECM    CHARSET=ALPHANUM,                                    X
               ASCCSID=1252,                                     X
               AMCCSID=65534,                                    X
               AGCCSID=65534,                                    X
               SCCSID=37,                                        X
               MCCSID=65534,                                     X
               GCCSID=65534,                                     X
               USCCSID=367,                                      X
               UMCCSID=1208,                                     X
               UGCCSID=1200,                                     X
               ENSCHEME=EBCDIC,                                  X
               APPENSCH=EBCDIC,                                  X
```

All character type data can be converted from the host internal code, predominantly from EBCDIC to other data types, such as ASCII and UNICODE on the S/390 DB2 databases. The UNLOAD statement in Example 3-9 converts all the character fields on table REGION into ASCII.

Example 3-9 Unload with character conversion to ASCII

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD,
//         UTSTATS=''
//SYSIN    DD  *
  TEMPLATE PNHDDN
```

```
    DSN(DB2V710G.RAMA.&TS..PUNCH)
    UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
    VOLUMES(SBOX57,SBOX60)
  TEMPLATE ULDDDN
    DSN(DB2V710G.RAMA.&TS..UNLOAD)
    UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
    VOLUMES(SBOX57,SBOX60)
  UNLOAD TABLESPACE U7G01T11.TSREGION
         PUNCHDDN PNHDDN UNLDDN ULDDDN ASCII NOPAD
```

The alternate way to convert to ASCII is to use the CCSID option in the UNLOAD, as follows:

```
UNLOAD TABLESPACE U7G01T11.TSREGION PUNCHDDN PNHDDN UNLDDN ULDDDN
CCSID(01252,00000,00000)
```

In Example 3-10, table PART in table space TSPART was created with CCSID 1140, Euro English code page. The table space TSPART was unloaded and CCSID was converted to UNICODE. Unload converted all character data from CCSID 1140 to 367.

Example 3-10 Unload with character conversion to UNICODE

```
//STEP1    EXEC DSNUPROC,SYSTEM=DB2G,UID=UNLOAD,
//         UTSTATS=''
//SYSIN    DD  *
  TEMPLATE PNHDDN
    DSN(DB2V710G.RAMA.&TS..PUNCH)
    UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
    VOLUMES(SBOX57,SBOX60)
  TEMPLATE ULDDDN
    DSN(DB2V710G.RAMA.&TS..UNLOAD)
    UNIT(SYSDA) CYL DISP(NEW,CATLG,DELETE)
    VOLUMES(SBOX57,SBOX60)
  UNLOAD TABLESPACE U7G01T11.TSPART
         PUNCHDDN PNHDDN  UNLDDN ULDDDN UNICODE
```

Note: Unload utilizes OS/390 services for CCSID conversion from 1140 to 367. There is no direct translation from 1140 to 367. Please refer to *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289 for examples of generating the translation table and installation of OS/390 UNICODE support.

Restrictions

- Unload is not supported for table spaces in DSNDB01.
- The FROM TABLE option cannot be used when the LIST option is already specified.
- The following table spaces and image copy source objects are not supported:
  - Auxiliary (LOB) table spaces
  - Index spaces
  - Dropped tables
  - Views
  - Global temporary tables

3.3.2 Terminating or restarting Unload

In this section we discuss terminating or restarting the Unload utility.

Terminating Unload

The Unload utility can be terminated with TERM UTILITY command during the Unload phase. The output records in SYSREC are not erased, but the data set remains incomplete until the data set is deleted or the Unload utility is restarted.

Restarting Unload

When restarting a terminated Unload utility, the following occurs:

- Processing starts with the table spaces or partitions that had not yet been completed.
- For a table space or partitions that were being processed at termination, Unload resets the output data sets and processes those table spaces or partitions again.
- When the source is one or more image copy data sets (FROMCOPY or FROMCOPYDDN), Unload always starts processing from the beginning.

Note: Verify that the PTF for APAR PQ50223 is applied to avoid a problem generated when a partitioned table space with PARTLEVEL in LISTDEF is unloaded with the Unload utility and loaded with the Load utility.

4

Load with DB2 for z/OS

In this chapter, we mainly discuss the new Cross Loader function, but only after giving a general description of the Load utility, of which Cross Loader is an option, as introduction.

These are the topics contained in this chapter:

- General description of the Load utility
- Sample JCL for the Load utility
- Some tips on using the Load utility
- General description of the Cross Loader function:
  - The INCURSOR option
  - EXEC SQL
  - Error behavior
  - Thread behavior
- Using the Cross Loader
- Cross Loader examples

4.1 The Load utility for DB2 for z/OS

In this section, we use the DB2 for z/OS Version 7 Load utility. This Load utility can load data into one or more DB2 tables. You can add data into an existing table or replace all existing data with new ones.

The authority needed to use the Load is any of the following:

- Ownership of the table
- Load privilege for the database
- DBADM or DBCTRL authority for the database
- SYSCTRL or SYSADM authority

Load will produce these outputs after its run:

- A loaded table space or partition
- A file containing rejected records
- A summary report of errors encountered during processing. This is generated only if you specify ENFORCE CONSTRAINTS or if the LOAD involves unique indexes.

With the current versions of DB2, you cannot Unload DB2 data into a delimited input file. A delimited file is a sequential file that contains row and column delimiters often used to move data across the distributed platforms. With the recently announced DB2 for z/OS Version 8, you will be able to use the Unload utility to unload DB2 data to one or more delimited output files. You can then load the data into another DB2 database in a z/OS environment or a distributed platform, or import the data into an application in another relational database. See the following Web site for more information on DB2 for z/OS Version 8:
http://www.ibm.com/software/data/db2/os390/db2zosv8.html

4.1.1 Input data for Load

There is some preparation you should do on your input data before performing the Load:

- Sort your data before doing the Load. Data will be loaded with the same sequence as it is found in the input file.
- Inspect the input and ensure that there are no duplicate keys for unique index.
- Correct check constraint violations and referential constraint violations in the input data set.
- When loading into a segmented table space, sort your data by table to ensure that the data is loaded in the best physical organization.

Table 4-1 summarizes the data sets used by the Load utility.

Table 4-1 Data sets used by Load

Data set	Description	Required
SYSIN	Input data set containing the utility control statement.	Yes
SYSPRINT	Output data set for messages	Yes
Input data set (INDD)	The input data set containing the data to be loaded. It must be a sequential data set readable by BSAM.	Yes

Data set	Description	Required
Sort data sets (SYSUT1) (SORTOUT)	These are temporary work data sets needed for the sort input and sort output. Their DD name is specified by the WORKDD option. The default DD name for the input is: SYSUT1 for the input and SORTOUT for the output.	Yes (required if referential constraints exist and ENFORCE (CONSTRAINTS) is specified or if the table has indexes.)
Mapping data set	Work data set for mapping the identifier of a table row back to the input record that caused an error. The default DD name is SYSMAP.	Yes (required if referential constraints exist and ENFORCE (CONSTRAINTS) is specified or when you request discard processing in loading tables with unique index.)
UTPRINT	Contains messages from DFSORT (usually, SYSOUT or DUMMY.)	No (required if SORT is done)
Discard data set	A work data set to hold copies of records not loaded. It must be a sequential data set that is readable by BSAM. Its DD or template name is specified with the DISCARDDN option of the utility control statement. If you omit this, Load creates the data set with the same record format, record length, and block size as the input data set. The default DD name is SYSDISC.	Yes (required if requested through the Discard options of the utility control statement.)
Error data set	Work data set for error processing. Its DD name is specified with the ERRDDN parameter of the utility control statement. The default DD or template name is SYSERR.	Yes
Copy data sets	1 to 4 output data sets to contain image copy data sets. Their DD or template names are specified with the COPYDDN and RECOVERYDDN options of the utility control statement.	No

For more information please refer to Chapter 2 of the *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference,* SC26-9945-03.

4.1.2 Sample Load JCL

These are some sample JCL that you can use to perform load operations. For more information on the options of the Load utility, please refer to *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference,* SC26-9945-03.

Example 1

Example 4-1 shows the JCL for loading a table with the RESUME YES. This job will place the table in the CHECK-pending status. The referential constraint will be enforced because this is the default unless you specify the ENFORCE NO option. See Example 4-1.

Example 4-1 Load JCL with RESUME YES

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='V72'
//SYSREC DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *
LOAD DATA INDDN(SYSREC) RESUME YES
INTO TABLE PAOLOR7.DEPT
   (DEPTNO    POSITION( 1) INTEGER EXTERNAL(4),
   MGRNME POSITION( 6) CHAR(7),
   DEPTDESC  POSITION(14) VARCHAR)
//*
```

Example 2

Example 4-2 specifies dynamic allocation of the required data sets by DFSORT, using the SORTDEVT and SORTNUM keywords. If sufficient virtual storage resources are available, one utility subtask pair will be started to build each index. This example does not require UTPRINnn DD statements, because it uses DSNUPROC to invoke utility processing, which includes a DD statement that allocates UTPRINT to SYSOUT. See Example 4-2.

Example 4-2 DFSORT dynamic allocation

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',UTPROC='',SYSTEM='DB2G7'
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT, DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(CYL,(10,20),,,ROUND)
//SYSERR   DD DSN=PAOLOR7.LOAD.SYSERR,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSMAP   DD DSN=PAOLOR7.LOAD.SYSMAP,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSREC   DSN=PAOLOR7.TEMP.DATA,DISP=SHR,UNIT=SYSDA
//SYSIN    DD *
   LOAD DATA REPLACE INDDN SYSREC CONTINUEIF(79:80)='++'
   SORTKEYS 66000 SORTDEVT SYSDA SORTNUM 3
   INTO TABLE PAOLOR7.DEPT
//*
```

Example 3

Example 4-3 shows how to load selected columns into a non-empty table space. The input sequential data file is in SYSREC DD.

Example 4-3 RESUME with selected columns

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='V72'
//SYSREC DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *
   LOAD DATA
   RESUME YES
   INTO TABLE PAOLOR7.DEPT WHEN (1:3)='LKA'
   (DEPTNO POSITION (7:9) CHAR,
   DEPTNAME POSITION (10:35) CHAR,
   MGRNM POSITION (36:41) CHAR,
   ADMRDEPT POSITION (42:44) CHAR)
//*
```

For each source record that has LKA in its first three positions:

- The characters in positions 7 through 9 are loaded into the DEPTNO column.
- The characters in positions 10 through 35 are loaded into the DEPTNAME VARCHAR column.
- The characters in positions 36 through 41 are loaded into the MGRNO column.
- Characters in positions 42 through 44 are loaded into the ADMRDEPT column.

Example 4

Example 4-4 shows how to load data into table PAOLOR7.DEPT from the data set specified by the DEPTDS DD statement.

Example 4-4 Loading from a data set

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='DB2G7'
//DEPTDS DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *
   LOAD DATA INDDN DEPTDS
   INTO TABLE PAOLOR7.DEPT
//*
```

Example 5

Example 4-5 shows how to load data into two tables. Load data from the data set specified by the DEPTDS DD statement into the PAOLOR7.DEPT and SANCHEZ.TESTDEPT tables.

Example 4-5 Loading into two tables

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='V72'
//DEPTDS DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *
   LOAD DATA INDDN DEPTDS
   INTO TABLE PAOLOR7.DEPT
   INTO TABLE SANCHEZ.TESTDEPT
//*
```

Example 6

Example 4-6 shows how to load ASCII input data. Use the ASCII option to load ASCII input data into a table named DEPT that was created with the CCSID ASCII clause.

Example 4-6 Loading ASCII input data

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='V72'
//DEPTDS DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *

   LOAD REPLACE LOG NO ASCII INTO TABLE DEPT
         (EMPNAME POSITION(1) CHAR(40),
          ADDRESS POSITON(41) CHAR(40),
         SALARAY POSITION(81) DECIMAL EXTERNAL(9),
          DEPARTMENT POSITION(90) CHAR(3),
          TITLE POSITION(93) GRAPHIC(20))
/*
```

The CCSID keyword is not specified in this example; therefore, the CCSIDs of the ASCII input data are assumed to be the ASCII CCSIDs specified at installation. Conversions are done only if the CCSIDs of the target table differ from the ASCII CCSIDs specified at installation.

Example 7

Example 4-7 shows how to load selected records into a table. Load data from the data set specified by the DEPTDS DD statement into the TESTDEPT table. Load only from source input records that begin with LKA.

Example 4-7 Loading selected input records

```
//JOBCARD JOB  ...
//STEP1    EXEC DSNUPROC,UID='PAOLOR7.LOAD',
//         UTPROC='',
//         SYSTEM='V72'
//DEPTDS DD DSN=PAOLOR7.LOAD.DATA,DISP=SHR,VOL=SER=DB2G7,
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSUT1   DD DSN=PAOLOR7.LOAD.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SORTOUT  DD DSN=PAOLOR7.LOAD.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(3500,(10,10),,,ROUND)
//SYSIN    DD *
LOAD DATA INDDN DEPTDS
INTO TABLE SANCHEZ.TESTDEPT
WHEN (1:3)='LKA'
//
```

4.1.3 Some tips on using the Load

The Load utility from the DB2 Version 7 has a lot of new functions. Major new usability features include the use of TEMPLATE and EXEC SQL. Major new functionality features include the Cross Loader and Online LOAD RESUME. Major performance improvements can be achieved by using Partition Parallelism, Inline COPY, Inline RUNSTATS and Parallel Index Build (PIB) by means of the SORTKEYS option.

Here are some recommendations:

- Always sort the input data in clustering sequence before loading. Do this with an ORDER BY clause when using the Cross Loader.
- Use templates whenever possible to allocate the Load work data sets dynamically.
- Use EXEC SQL to execute dynamic SQL statements before, between, or after utility statements; and to simplify the JCL of the utility jobs. Bear in mind that the use of EXEC SQL is not limited to the Load utility, but can be used with any utility.
- Use the Cross Loader to load data from any DRDA source, eliminating unloading and file transfers.
- Use Online LOAD RESUME to load data if concurrent access from applications is needed.
- Use LOAD REPLACE with LOG NO and Inline COPY and inline statistics.
- Use LOAD RESUME NO with inline statistics.
- Use PIB by specifying SORTKEYS n to have your indexes built in parallel; n being an estimation of the number of keys to be sorted.
- Use multiple input data sets to load partitions in parallel, reducing elapsed load times and NPI contention.

4.2 Cross Loader option

The Cross Loader is a new capability in DB2 V7 that allows you to transfer data from one location to another location within a single utility job. Refer to Figure 4-1.

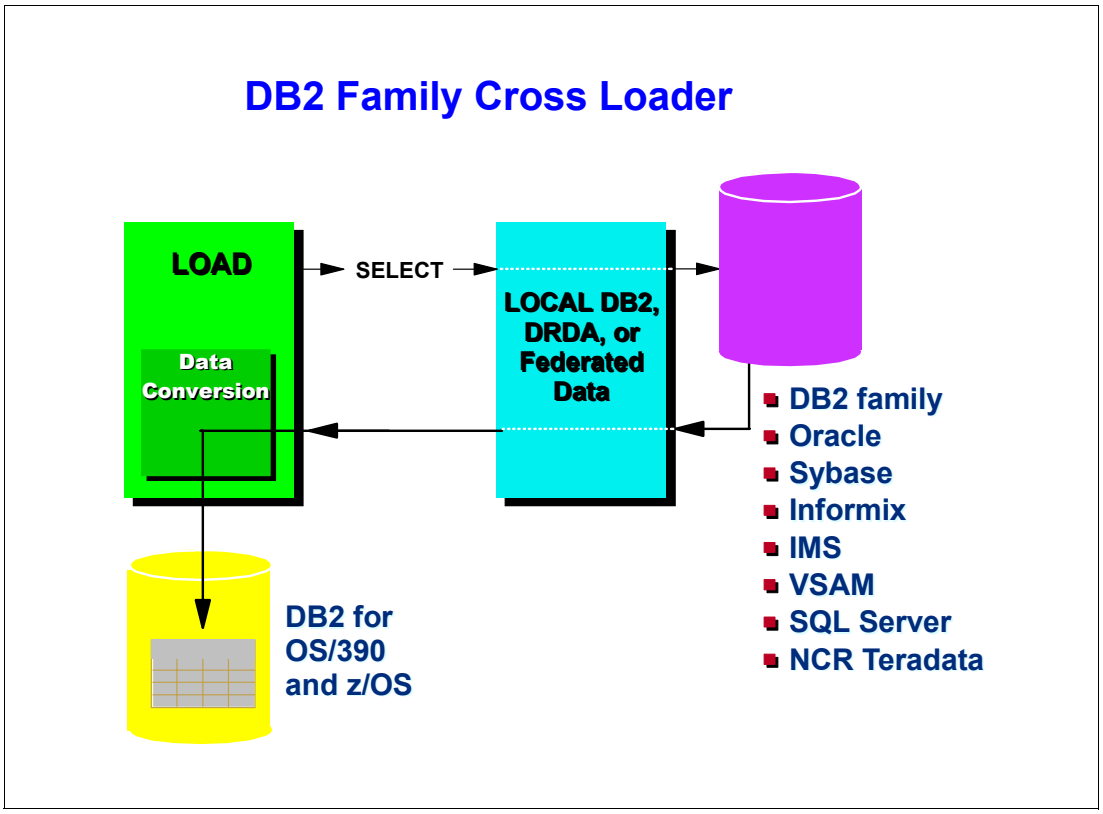

Figure 4-1 DB2 Family Cross Loader

The data is read from the source location with a dynamic SQL statement and loaded in a table at the target location by the Load utility. It is a single job process that replaces the typical sequence of jobs of unloading, file transfer, and loading the data. The source data can be on the local system or on a remote DRDA server.

Note: The source can be on any system accessible via a Federated Database, however this book will focus only on the DB2 Family.

A typical Cross Loader example consists of the definition of the dynamic SQL statement via the EXEC SQL DECLARE CURSOR utility statement, followed by a Load utility statement referring to this cursor. This is illustrated in Example 4-8. In this example we load an existing PAOLOR2.DEPT table with data from the local sample PAOLOR7.DEPT.

Example 4-8 Introductory Cross Loader example

```
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT *
  FROM PAOLOR7.DEPT
ENDEXEC

LOAD DATA
INCURSOR C1
```

```
INTO TABLE
PAOLOR2.DEPT
```

EXEC SQL is a new DB2 V7 utility statement explained in more detail in 4.2.2, “EXEC SQL utility control statement” on page 59.

INCURSOR is a new DB2 V7 LOAD option explained in more detail in 4.2.1, “INCURSOR Load option” on page 59.

Important: The Cross Loader was introduced in DB2 V7 after GA with PTF UQ55541 for APAR PQ45268 and PTF UQ55542 for APAR PQ46759. Therefore, you should check if these PTFs, and all their prerequisites, are applied to your DB2 system before trying to use the Cross Loader.

The Cross Loader is being made available to all members of the DB2 Family. Once implemented it allows the source data to reside on any remote system that can act as a DRDA server.

The following aspects is discussed about the Cross Loader:

- “INCURSOR Load option” on page 59
- “EXEC SQL utility control statement” on page 59
- “Using the Cross Loader” on page 65

4.2.1 INCURSOR Load option

Instead of INDDN *ddname* you can specify the INCURSOR *cursor-name* for the input data to the Load utility:

- Use the EXEC SQL to define the cursor for the Cross Loader function.
- The cursor must be declared before it is used by the Load utility.
- The column names in the SELECT statement must be identical to the column names in the table being loaded.
- You cannot load into the same table where you defined the cursor.
- You can use the same cursor to load multiple tables.
- The INCURSOR option is incompatible with the SHRLEVEL CHANGE Load option.

To match the column names in the table being loaded, you can use the AS clause in the select list to change the columns names returned by the SELECT statement.

4.2.2 EXEC SQL utility control statement

DB2 V7 gives you the ability to execute dynamic SQL statements within the utility input stream. This is done via the new EXEC SQL utility control statement.

The purpose of using SQL statements in a utility input stream

EXEC SQL can be placed anywhere in the utility input stream. It can be used for two purposes:

- Executing a non-select dynamic SQL statement before, between or after the actual utility statements
- Declaring a cursor with a SQL select statement for use with the Load utility (Cross Loader). The declare cursor must be declared before the LOAD statement and it produces a result set to be loaded.

The EXEC SQL statement requires no extra privileges other than the ones required by the SQL statements itself.

Executing a non-select dynamic SQL statement

Executing a non-select dynamic SQL statement is done by putting it between the EXEC SQL and ENDEXEC keywords in the utility input stream:

```
EXEC SQL non-select dynamic SQL statement ENDEXEC
```

You can only put one SQL statement between the EXEC SQL and ENDEXEC keywords. The SQL statement can be any dynamic SQL statement that can be used as input for the EXECUTE IMMEDIATE statement:

- CREATE, ALTER, DROP a DB2 object
- RENAME a DB2 table
- COMMENT ON, LABEL ON a DB2 table, view, or column
- GRANT, REVOKE a DB2 authority
- DELETE, INSERT, UPDATE SQL operation
- LOCK TABLE operation
- EXPLAIN a SQL statement
- SET CURRENT register
- COMMIT, ROLLBACK operation

(see “Thread behavior and commit scope of the EXEC SQL utility statement” on page 62)

In Example 4-9 we create a new table in the default database DSNDB04 with the same layout as PAOLOR7.DEPT.

Example 4-9 Creating a new table using LIKE

```
EXEC SQL
  CREATE TABLE PAOLOR2.DEPT LIKE PAOLOR7.DEPT
ENDEXEC
```

In Example 4-10 we give this table a comment and allow everybody read access. Note the two separate steps.

Example 4-10 Comment and grant in two separate EXEC SQL steps

```
EXEC SQL
  COMMENT ON TABLE PAOLOR2.DEPT IS 'Copy of Paolor7 DEPT'
ENDEXEC
EXEC SQL
  GRANT SELECT ON PAOLOR2.SYSTABLES TO PUBLIC
ENDEXEC
```

In the same way, we are able to create indexes on this table, create views on it, and so on. All this is done in the utility input stream.

Declaring a cursor for use with the Cross Loader

The cursor definition has to be put between the EXEC SQL and ENDEXEC keywords in the utility input stream:

```
EXEC SQL DECLARE cursor-name CURSOR FOR select statement ENDEXEC
```

In Example 4-11 we declare a cursor for extracting rows with ADMRDEPT = ‘A00’ from PAOLOR7.DEPT.

Example 4-11 Declare a cursor for the Cross Loader

```
EXEC SQL
  DECLARE C1 CURSOR FOR
```

```
  SELECT * FROM PAOLOR7.DEPT
  WHERE ADMRDEPT = 'A00'
  ORDER BY 1 ASC
ENDEXEC
```

In Example 4-12 we show how this cursor can now be used in a LOAD statement (Cross Loader.)

Example 4-12 Usage of a cursor in the LOAD statement

```
LOAD DATA
INCURSOR C1
INTO TABLE PAOLOR2.DEPT
```

The SQL statement in the declare cursor definition can be any valid SQL statement including joins, unions, data conversions, aggregations, special registers, and UDFs. The source data can be on a local server or remote server using DRDA access. See 4.2.3, “Using the Cross Loader” on page 65 for more details.

Error behavior and restartability of the EXEC SQL utility statement

The EXEC SQL utility statements are executed in a new utility phase called the EXEC phase.

The SQL statement placed after the EXEC SQL keyword is parsed and checked for errors during its execution. It is not checked during the UTILINIT phase of the utility. If an invalid SQL statement is found during the execution of the EXEC SQL, the utility job immediately ends with return code 8. If the SQL statement is valid, but fails during execution (with a negative SQL code), the utility also immediately ends with return code 8 as well. So be aware that if you have syntax errors in an EXEC SQL statement and the utility job gets started, the previous EXEC SQL statements and utility statements are already executed before the utility ends. You might have to remove these from the input stream before rerunning the job.

Normally, a utility that encounters an SQL error during the EXEC SQL statement execution always ends with return code 8 and never abends with ABEND04E. So the utility is not in a stopped state; the utility is not restartable with the RESTART option and a *TERMINATE UTILITY* command is NOT necessary. But be aware that all previous EXEC SQL and utility statements are executed successfully and might have to be removed first before rerunning the job.

Currently, it is impossible to influence the behavior of the utility job after a failing EXEC SQL statement. The OPTION to allow to discard the failing EXEC SQL, and to continue the utility step when the EXEC SQL failed, is currently not available in DB2 V7.

If the utility input stream contains both EXEC SQL statements and other utility statements, and a utility statement failed so that DB2 put the utility in the stopped state, the utility step is restartable with the RESTART keyword. During restart, all the non-select dynamic SQL statements from EXEC SQL statements already executed, are skipped, except the ones with SQL SET statements. All the declare cursor definitions within EXEC SQL statements already executed, are reactivated so that they can be used in the following LOAD statements.

This can be illustrated with Example 4-13. This job contains one non-select dynamic SQL statement (*CREATE TABLE*), one cursor definition, and one utility LOAD statement. If the CREATE TABLE fails with a negative SQL code, the utility will immediately end with return code 8 and the utility will not be restartable with the RESTART keyword. If the CREATE TABLE executes successfully, but the DECLARE CURSOR fails, the utility will also end with return code 8, but the table will have been created. If both CREATE TABLE and DECLARE

CURSOR execute successfully, but the LOAD statement fails so that DB2 puts the utility in the stopped state (for example, because of a resource unavailable condition) the utility will be restartable. During restart the CREATE TABLE statement will be skipped, but the DECLARE CURSOR statement will be re-executed so that it can be used in the LOAD statement.

Example 4-13 Restartability of the EXEC SQL statement

```
EXEC SQL
  CREATE TABLE DEPTCOPY LIKE DEPT
ENDEXEC
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT * FROM DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  INTO TABLE DEPTCOPY
```

Thread behavior and commit scope of the EXEC SQL utility statement

The EXEC SQL statement runs in a thread that is separate from the utility thread. This implies that the EXEC SQL SET CURRENT register operations do influence all following EXEC SQL statements in the utility stream input, but they do not influence the real utility statements like LOAD, REORG, and so on.

We can illustrate this with the utility job in Example 4-14. This job runs with USER=PAOLOR2. So, the DB2 primary AUTHID is PAOLOR2. We change the current SQLID with EXEC SQL to PAOLOR7 and try to see what table creator is used if we do not prefix our tables in the following EXEC SQL statements and other utility statements. We only want to load the 100 most recently created tables. In V7 we can do this using the new SQL fetch-first-clause.

Example 4-14 JCL for testing the thread behavior of EXEC SQL

```
//PAOLOR2A JOB (ACCOUNT),'PAOLOR2',NOTIFY=PAOLOR2,USER=PAOLOR2
//*
// EXEC PGM=DSNUTILB,PARM='DB2G,CRLOAD'
//STEPLIB DD DSN=DSN710.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  EXEC SQL
    SET CURRENT SQLID = 'PAOLOR7'
  ENDEXEC
  EXEC SQL
    CREATE TABLE MYTABLE LIKE SYSIBM.SYSTABLES
  ENDEXEC
  EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT * FROM SYSIBM.SYSTABLES
    WHERE TYPE = 'T'
    ORDER BY CREATEDTS DESC
    FETCH FIRST 100 ROWS ONLY
  ENDEXEC
  LOAD DATA
  INCURSOR C1
  INTO TABLE PAOLOR7.MYTABLE
  LOAD DATA RESUME(YES)
  INCURSOR C1
  INTO TABLE         MYTABLE
```

The resulting job output is shown in Example 4-15.

In the first EXEC SQL, the current SQLID is set to PAOLOR7. In the following EXEC SQL, the table MYTABLE is created without specifying a table creator. We verified in the DB2 catalog that this table is created with CREATOR = PAOLOR7, which proves that the previous current SQLID was used in this EXEC SQL. If we try to load this table with the Cross Loader, we have to specify PAOLOR7.MYTABLE, otherwise, the load fails because the Load utility thread does not use the current SQLID set in the previous EXEC SQL. Instead, its current SQLID is still equal to the primary AUTHID, PAOLOR2, which it inherited from the USER keyword in the JCL.

Example 4-15 Testing the thread behavior of EXEC SQL

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CRLOAD
DSNU050I    DSNUGUTC -  EXEC SQL SET CURRENT SQLID='PAOLOR7' ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU050I    DSNUGUTC -  EXEC SQL CREATE TABLE MYTABLE LIKE SYSIBM.SYSTABLES ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU050I    DSNUGUTC -  EXEC SQL DECLARE C1 CURSOR FOR SELECT * FROM SYSIBM.SYSTABLES WHERE TYPE='T' ORDER BY
CREATEDTS DESC FETCH FIRST 100 ROWS ONLY ENDEXEC
DSNU050I    DSNUGUTC -  LOAD DATA INCURSOR C1
DSNU650I  -DB2G DSNURWI -  INTO TABLE PAOLOR7.MYTABLE
DSNU304I  -DB2G DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=100 FOR TABLE PAOLOR7.MYTABLE
DSNU302I    DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=100
DSNU300I    DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU050I    DSNUGUTC -  LOAD DATA RESUME(YES) INCURSOR C1
DSNU650I  -DB2G DSNURWI -  INTO TABLE MYTABLE
DSNU056I  -DB2G DSNUGMAP - TABLE 'PAOLOR2.MYTABLE' NOT FOUND
DSNU012I    DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

Each block of EXEC SQL and ENDEXEC is a separate unit-of-work. Each block can only contain a single SQL statement. The unit-of-work is always committed when the SQL statement is executed successfully. Although the COMMIT and ROLLBACK statements are accepted, these statements do not perform any function.

So, it is impossible to create your own unit-of-work consisting of multiple EXEC SQL statements, and end that unit-of-work with EXEC SQL COMMIT ENDEXEC. It is also impossible to have an EXEC SQL statement executed and undo its work by a following EXEC SQL ROLLBACK ENDEXEC command.

We verified the above behavior with the utility job shown in Example 4-16. In the first EXEC SQL we create a new table. This is followed by an EXEC SQL ROLLBACK to try to undo the CREATE statement. In the third EXEC SQL we populate this table with an INSERT statement and try to undo the INSERT in the fourth EXEC SQL. If the ROLLBACK statement in the second EXEC SQL would undo the CREATE, we expect the third EXEC SQL to fail.

Example 4-16 JCL for verifying the commit scope of EXEC SQL

```
//PAOLOR2A JOB (ACCOUNT),'PAOLOR2',NOTIFY=PAOLOR2,USER=PAOLOR2
// EXEC PGM=DSNUTILB,PARM='DB2G,CRLOAD'
//STEPLIB DD DSN=DSN710.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  EXEC SQL
    CREATE TABLE PAOLOR2.SYSTABLESR LIKE SYSIBM.SYSTABLES
  ENDEXEC
  EXEC SQL
    ROLLBACK
  ENDEXEC
  EXEC SQL
    INSERT INTO PAOLOR2.SYSTABLESR
    SELECT * FROM SYSIBM.SYSTABLES
    WHERE TYPE = 'T'
      AND CREATOR LIKE 'PAOLOR%'
  ENDEXEC
  EXEC SQL
```

```
   ROLLBACK
ENDEXEC
```

The result of this job is shown in Example 4-17. As you can see, all four EXEC SQL statements executed successfully. We verified with SPUFI that the table PAOLOR2.SYSTABLESR was successfully created and populated with 35 rows. So, the EXEC SQL ROLLBACK statements did not influence the previous EXEC SQL statements.

Example 4-17 Verifying the commit scope of EXEC SQL

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = CRLOAD
DSNU050I    DSNUGUTC -  EXEC SQL CREATE TABLE PAOLOR2.SYSTABLESR LIKE SYSIBM.SYSTABLES ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU050I    DSNUGUTC -  EXEC SQL ROLLBACK ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU050I    DSNUGUTC -  EXEC SQL INSERT INTO PAOLOR2.SYSTABLESR SELECT * FROM SYSIBM.SYSTABLES WHERE TYPE='T' AND
CREATOR LIKE 'PAOLOR%' ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU1196I   DSNUGSQL -  SQLERRD    = 0  0  35  1127790002  0  0 SQL DIAGNOSTIC INFORMATION
DSNU1196I   DSNUGSQL -  SQLERRD    = X'00000000'  X'00000000'  X'00000023'  X'4338B5B2'  X'00000000'  X'00000000' SQL
                        DIAGNOSTIC INFORMATION
DSNU050I    DSNUGUTC -  EXEC SQL ROLLBACK ENDEXEC
DSNU1180I   DSNUGSQL -  SQLCODE = 000,  SUCCESSFUL EXECUTION
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

So, although you can code COMMIT and ROLLBACK statements, after the EXEC SQL keyword, they do not influence the behavior of previous EXEC SQL commands.

Possible usage of the EXEC SQL utility statement

The primary use of the EXEC SQL utility statement is meant for declaring cursors for use with the Load utility (Cross Loader). But it can also be used to execute any non-select dynamic SQL statement before, between or after regular utility statements. Examples are:

- DDL creation of the target table for the Load utility (and its related objects like database, table space, indexes, views)
- DDL creation of the mapping table and index before a REORG table space SHRLEVEL CHANGE
- Dropping of the mapping table after a successful REORG table space SHRLEVEL CHANGE
- DDL alter of space related values like PRIQTY, SECQTY, FREEPAGE and PCTFREE values before a Reorg or Load utility
- DDL alter of INDEX partitions before Reorg (for partitioning key changes)
- GRANT statements (example: grant select authorities after successful load)
- SQL delete of *old* records before LOAD RESUME YES
- SQL update of an application related control table or SQL insert into an application related control table after successful LOAD (with the current timestamp)

The benefits of EXEC SQL are:

- You can execute any non-select dynamically prepared SQL statement within the utility input stream.
- You can declare cursors for use with the Load utility, including joins, unions, conversions, aggregations, and remote DRDA access.
- Successfully executed SQL statements are skipped during restart of the utility.
- In many cases, the need for extra dynamic SQL programs in the utility job stream is eliminated.

- Considerable simplification of JCL is possible.

Be aware that there are restrictions imposed by the EXEC SQL statement:

- There are no select statements.
- There is no control after error; the whole utility step stops after the first SQL error.
- There is no concept of unit-of-work consisting of multiple SQL statements.
- There are no comments possible between SQL statements.

4.2.3 Using the Cross Loader

As mentioned, two steps are necessary to use the Cross Loader:

1. Declare a cursor with the EXEC SQL statement.
2. Load the data into the target table using above cursor.

However, some customizing steps have to be completed before you can use Cross Loader to load data.

Binding the Cross Loader package

The Cross Loader uses package *DSNUGSQL* in collection *DSNUTIL* that must be bound local and at the remote DRDA servers you want to transfer data from. Locally, it must be bound with the option DBPROTOCOL(DRDA) to allow the DRDA protocol to use three-part-names. It uses the default system utility plan DSNUTIL. The bind command on the local system is done in installation job DSNTIJSG.

Cross Loader package on local system

You can verify the existence of a valid and operative DSNUGSQL; see Example 4-18.

Example 4-18 Local Cross Loader to verify DSNUGSQL package

```
EXEC SQL
  CREATE TABLE MY.SYSTABLES LIKE SYSIBM.SYSTABLES
ENDEXEC
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT * FROM SYSIBM.SYSTABLES
ENDEXEC
LOAD DATA
INCURSOR C1
INTO TABLE MY.SYSTABLES
```

If the package for any reason is not available, you have to bind the package, as shown in Example 4-19.

Example 4-19 Local bind of DSNUGSQL package

```
BIND PACKAGE(DSNUTIL) MEMBER(DSNUGSQL)               -
     ACTION(REPLACE) ISOLATION(CS) ENCODING(EBCDIC)  -
     VALIDATE(BIND) CURRENTDATA(NO)  KEEPDYNAMIC(NO) -
     LIBRARY('DB2G7.SDSNDBRM')
```

The result of the bind, see Example 4-20.

Example 4-20 Successful local bind of DSNUGSQL with options

```
DSNT254I  -DB2G DSNTBCM2 BIND OPTIONS FOR
           PACKAGE = DB2G.DSNUTIL.DSNUGSQL.(V7R1)
           ACTION        REPLACE
           OWNER         PAOLOR2
           QUALIFIER     PAOLOR2
           VALIDATE      BIND
           EXPLAIN       NO
           ISOLATION     CS
           RELEASE
           COPY
DSNT255I  -DB2G DSNTBCM2 BIND OPTIONS FOR
           PACKAGE = DB2G.DSNUTIL.DSNUGSQL.(V7R1)
           SQLERROR      NOPACKAGE
           CURRENTDATA   NO
           DEGREE        1
           DYNAMICRULES
           DEFER
           NOREOPT       VARS
           KEEPDYNAMIC   NO
           IMMEDWRITE    NO
           DBPROTOCOL    DRDA
           OPTHINT
           ENCODING      EBCDIC(00037)
           PATH
DSNT232I  -DB2G SUCCESSFUL BIND FOR
           PACKAGE = DB2G.DSNUTIL.DSNUGSQL.(V7R1)
```

Remember to grant privileges on the Cross Loader package:

```
GRANT EXECUTE ON PACKAGE DSNUTIL.DSNUGSQL TO PUBLIC
```

Cross Loader package on remote system

You have to bind the package on the remote DRDA system, as shown in Example 4-21.

Example 4-21 Remote bind of DSNUGSQL package

```
BIND PACKAGE (SAMPAIX.DSNUTIL)                    -
              COPY(DSNUTIL.DSNUGSQL) COPYVER(V7R1) -
              CURRENTDATA(NO) ISOLATION(CS)        -
              ACTION(REPLACE) OPTIONS(COMMAND)
```

See the result of the bind in Example 4-22. Make sure that CURRENTDATA is set to NO. If not, it will have a negative influence on the data transmission between the remote and the local server.

Example 4-22 Successful remote bind of DSNUGSQL

```
   BIND PACKAGE (SAMPAIX.DSNUTIL) COPY(DSNUTIL.DSNUGSQL) COPYVER(V7R1)
   CURRENTDATA(NO) ISOLATION(CS) ACTION(REPLACE) OPTIONS(COMMAND)
WARNING, ONLY IBM-SUPPLIED COLLECTION-IDS SHOULD BEGIN WITH  "DSN"
WARNING, ONLY IBM-SUPPLIED COLLECTION-IDS SHOULD BEGIN WITH  "DSN"
WARNING, ONLY IBM-SUPPLIED PACKAGE-IDS SHOULD BEGIN WITH  "DSN"
DSNT232I  -DB2G SUCCESSFUL BIND FOR
           PACKAGE = SAMPAIX.DSNUTIL.DSNUGSQL.(V7R1)
```

You have to do a remote bind on all remote servers involved in the cross loading. In this redbook project this means both SAMPAIX and SAMPWIN. See appendix B.

Important: If you get the following error when loading from a cursor declared on a table residing on an UNIX server:

```
UTILITY BATCH MEMORY EXECUTION ABENDED, REASON=X'00E4D5D2'
```

make sure that PTFs for PQ67037 and PQ62837 are applied to your DB2 system.

Rules for declaring a cursor for use with the Cross Loader

The following rules apply to the cursor you DECLARE in the EXEC SQL statement:

- You must always declare the cursor *before* the LOAD statement. Otherwise, you get the message *DSNU1190I CURSOR NOT DECLARED.*
- The cursor name can only be *declared once* within the whole utility input stream. It can be referred in multiple LOAD statements for LOADING different target tables with the same data. If you declare an existing cursor, you get the error message: *DSNU1189I CURSOR ALREADY DECLARED.*
- The cursor name can only have up to eight characters.
- The table being loaded cannot be part of the select statement. So you cannot LOAD into the same table where you defined the cursor.
- The column names in the result set of the select statement must be *identical* to the column names in the table being loaded. This can be achieved by using the *AS clause* in the SQL statement. If you have column names in the result set which do not match any column name in the target table you get the error message: *DSNU053I FIELD 'colname' NOT FOUND* or the error message: *DSNU329I FIELD 'colnumber' IS NOT DEFAULTABLE.*
  Pay special attention to derived column names, which are the result of column functions such as SUM or AVG.
- You are able to skip unwanted columns in the result set with the *LOAD IGNOREFIELDS YES* option, which skips any columns in the cursor result set that are not present in the target table being loaded. However, use this IGNOREFIELDS option with care, as it also skips misspelled columns you wanted to LOAD.
- The sequence of the columns in the result set *may differ* from the sequence of the columns in the table being loaded. DB2 matches the columns by their names and not by their sequence.
- The number of columns in the cursor can be less than the number of columns in the target table. All missing columns are loaded with their *default values*. If the missing columns are defined in the target table as NOT NULL without default, the LOAD fails with this message: *DSNU323I COLUMN 'colname' IS OMITTED.*
- If the data types in the target table do not match with the data types in the cursor, DB2 tries to *convert* as much as possible between compatible data types. Examples are from CHAR to VARCHAR or from INTEGER to FLOAT. If the conversion fails, you get messages like: *DSNU390I INVALID CONVERSION FOR FIELD 'columnname'* (conversion error detected before the actual LOAD during the matching process) or *DSNU334I INPUT FIELD 'columnname' INVALID FOR 'tablename', ERROR CODE cc* (conversion error detected during the LOAD). You might use DB2 supplied built-in functions or own developed UDFs in the SQL statement to force more sophisticated conversions. An example is the CHAR function which allows you to convert from INTEGER to CHARACTER.
- If the *encoding scheme* of the source data in the cursor and the target table differ, DB2 automatically converts the encoding schemes. An example may be conversion from EBCDIC to UNICODE or from ASCII to EBCDIC. Remember that referencing tables with

more than one encoding scheme (ASCII,EBCDIC or UNICODE) in a single SQL statement is not supported and finishes with SQLCODE -873.

- If the target table contains UDTs, you do not have to specify the appropriate casting functions in the cursor SQL statement to LOAD the table. If the source data in the cursor contains UDTs, you also do not have to specify casting functions in the select list of the cursor SQL statement. But additional WHERE clauses in the cursor SQL statement might require casting functions, or you could end up with SQLCODE -401.
- If your table contains LOB columns, the maximum length is *32 KB*. You cannot use the Cross Loader if you want to transfer LOB columns larger than 32 KB. Instead you should use a program with embedded SQL. If a table contains LOB columns, there is at least one ROWID column. If this ROWID column is created with the GENERATED ALWAYS clause (the recommended default) you cannot specify this column in the select list of the cursor. Instead DB2 generates the target ROWID value during loading. If you do specify the ROWID column in the select list you get message: *DSNU269I FIELD columnname IS NOT ALLOWED*. If the ROWID column is created with the GENERATED BY DEFAULT clause, you may specify this column in the select list. In this case the source ROWID value is copied. Do not forget to create a unique index on the ROWID column if you specified GENERATED BY DEFAULT or you fail with error message: *DSNU309I NOT ALL REQUIRED UNIQUE INDEXES HAVE BEEN DEFINED FOR TABLE tablename*.
- Apart from the aforementioned rules, the SQL statement in the declare cursor definition can be any valid SQL statement including joins, unions, conversions, aggregations, special registers, UDFs. The source data can be *local* or *remote* or a nickname in a *Federated Database* using *DRDA* access. Remote tables are always referred by their three-part-name *LOCATION.CREATOR.NAME* or by an ALIAS name *CREATOR.NAME.* You cannot use an explicit CONNECT statement to connect to the remote location.

Load the data into the target table using a cursor

Cross loading is done using the new option *INCURSOR cursor*. With this option you tell the Load to get the data from the result set of the cursor instead of getting it from a sequential data set referred by the DD statement or template INDDN (with default SYSREC.)

You can use any option of the Load utility except the following:

- **SHRLEVEL CHANGE:** There is no support for online Load in the Cross Loader. If you specify SHRLEVEL CHANGE, you get the message: *DSNU070I KEYWORD OR OPERAND 'SHRLEVEL CHANGE' INVALID WITH INCURSOR*.
- **FORMAT:** You cannot specify the UNLOAD or SQL/DS formats.
- **FLOAT(IEEE):** The cursor always returns FLOAT(S390.)
- **EBCDIC, ASCII, UNICODE:** The Cross Loader always automatically converts the encoding schemes. The target table must be created with the correct CCSID.
- **NOSUBS:** You must accept substitution characters in a string.
- **CONTINUEIF:** You cannot treat each input record as a part of a larger record.
- **WHEN:** You cannot use the WHEN option to filter the result set of the cursor. Instead, filter the result set by using the appropriate WHERE clauses in the select statement.
- **Field specifications:** You cannot specify field specifications in the LOAD Statement. If you specify field specifications, you get the message: *DSNU331I FIELD LISTS ARE NOT ALLOWED WITH INCURSOR KEYWORD*.

The same cursor can be reused multiple times in the utility job step. It can be referenced in different LOAD statements to load the same data in different target tables. It can also be reused in one LOAD statement containing multiple INTO TABLE clauses to LOAD the same data in different target tables (in the same table space.)

If a cursor is used more than once in the utility job step, the data is transferred more than once as well. Each time you refer to a cursor name in a LOAD statement, the SQL statement is re-executed. There is no buffering nor reuse of the previous transferred data. So the result sets can differ if the source data is being updated or if you use time dependent functions like CURRENT TIMESTAMP.

It is recommended that you load your data in *clustering sequence*. If loading from a sequential data set this can be done by first sorting the sequential data set in clustering sequence. With the Cross Loader this can be achieved by sorting the result set of the cursor by using an ORDER BY statement on the clustering columns.

You can load partitions in *parallel* by specifying a different cursor for each partition.

Cross Loader examples with local source and local target

The following examples is based on the tables PAOLOR2.DEPT and PAOLOR7.DEPT. The columns, their attributes and the order are the same:

```
   DEPTNO   CHAR(3)     NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO    CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   LOCATION CHAR(16))
```

Example 4-23 is a simple example of using the Cross Loader, where the columns in the cursor match exactly the columns in the target table (same column names, same column types, same order.)

Example 4-23 Cross Loader example with no columns referred

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT * FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
```

Instead of *, in Example 4-24, we are using the column names in the select list.

Example 4-24 Cross Loader example with column names

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, LOCATION
  FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-25 we are loading with the same column names, but in a different order.

Example 4-25 Cross Loader example with columns in different order

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, ADMRDEPT, LOCATION, MGRNO
  FROM PAOLOR7.DEPT
```

```
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-26 we are loading only with columns that are not defined as default. If you omit non-defaultable columns, the Load will terminate with:

```
   COLUMN column name IS OMITTED
```

Example 4-26 Cross Loader example with non-default columns

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT DEPTNO, DEPTNAME, ADMRDEPT
  FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-27 we show how the SQL AS clause can be used to match the column names in the cursor with the column names in the target table. You should use the SQL AS clause for every derived column name that is the result of columns functions (SUM,MAX,..) or UDFs.

Example 4-27 Cross Loader example with AS clause in the column list

```
EXEC SQL
  CREATE TABLE PAOLOR2.XDEPT
   (XDEPTNO   CHAR(3)     NOT NULL,
    XDEPTNAME VARCHAR(36) NOT NULL,
    XMGRNO    CHAR(6),
    XADMRDEPT CHAR(3)     NOT NULL,
    XLOCATION CHAR(16))
ENDEXEC
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT DEPTNO   AS XDEPTNO,
         DEPTNAME AS XDEPTNAME,
         MGRNO    AS XMGRNO,
         ADMRDEPT AS XADMRDEPT,
         LOCATION AS XLOCATION
  FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.XDEPT
```

In Example 4-28 we show the use of the IGNOREFIELDS YES LOAD option to ignore columns in the cursor that are not present in the target table.

Example 4-28 Cross Loader example with IGNOREFIELDS

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT DEPTNO,
         DEPTNAME,
         MGRNO,
```

```
        ADMRDEPT,
        LOCATION,
        CURRENT DATE AS DATE
  FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
  IGNOREFIELDS YES
```

To test automatic conversion between encoding schemes, we declared three different tables on the mainframe, see Example 4-29.

Example 4-29 The test tables used in Example 4-30

```
TS     TS       ENCODING   SBCS     DBCS     MIXED  TB       TB       ENCODING
NAME   CREATOR  SCHEME     CCSID    CCSID    CCSID  CREATOR  NAME     SCHEME
------ -------- -------- -------- -------- -------- -------- -------- --------
EMP    PAOLOR7  E             37        0        0 PAOLOR7  EMP      E
TSEMPA PAOLOR2  A           1252        0        0 PAOLOR2  EMPA     A
TSEMPE PAOLOR2  E             37        0        0 PAOLOR2  EMPE     E
TSEMPU PAOLOR2  U            367     1200     1208 PAOLOR2  EMPU     U
```

Example 4-30 shows the Cross Loading from one local EBCDIC table (PAOLOR7.EMP) to the tables in Example 4-29.

Example 4-30 Cross Loader conversion within the mainframe

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT * FROM PAOLOR7.EMP
  ORDER BY 1 DESC
ENDEXEC
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPE
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPA
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPU
```

Note: Make sure that the UNICODE translation table is activated with the necessary translation combinations on the mainframe.

We will now LOAD a table containing UDT columns. In Example 4-31 we first create and populate a table containing a UDT column. This UDT is defined as *CHAR(3) WITH COMPARISONS*. We do not have to specify the appropriate casting function in the select list of the cursor SQL statement to load the table.

Example 4-31 Cross Loader example of populating a table with UDT column

```
EXEC SQL
  CREATE DISTINCT TYPE PAOLOR2.DEPT_OWNER AS CHAR(3) WITH COMPARISONS
ENDEXEC
EXEC SQL
```

```
  CREATE TABLE PAOLOR2.YDEPT
   (DEPTNO   PAOLOR2.DEPT_OWNER,
    DEPTNAME VARCHAR(36) NOT NULL,
    MGRNO    CHAR(6),
    ADMRDEPT CHAR(3)     NOT NULL,
    LOCATION CHAR(16))
ENDEXEC
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM PAOLOR7.DEPT
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.YDEPT
```

In Example 4-32 we copy a subset of this table back to PAOLOR3.EXAMP3 with the Cross Loader, converting the UDT back to the standard CHAR data type. We do not have to specify any casting function in the select list of the cursor but we have to specify it in the WHERE clause to create the subset. We only want to copy the rows corresponding with COL1 equals to ‘SYSIBM’ and rename the CREATOR value to ‘SYSIBMX’ to prevent duplicate values in PAOLOR3.EXAMP3. As we cannot use an Inline COPY with LOAD RESUME(YES), we have to take the image copy with an additional COPY statement.

Example 4-32 Cross Loader example using a table with UDTs as source

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT
    'AOO' AS DEPTNO,
    DEPTNAME,
    MGRNO,
    ADMRDEPT,
    LOCATION
  FROM PAOLOR2.YDEPT
  WHERE DEPTNO = PAOLOR2.DEPT_OWNER('AOO')
ENDEXEC
LOAD DATA
  INCURSOR C1
  RESUME YES
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-33 we populate a table containing a **LOB column** with the Cross Loader. Remember that the maximum LOB size that can be transferred by the Cross Loader is 32 KB. We first create the target base table, auxiliary table, and corresponding table spaces and indexes with the EXEC SQL statement. We declare the ROWID as NOT NULL GENERATED ALWAYS and do not copy the ROWID value from the source table. After the LOAD we collect STATISTICS on all table spaces using a LISTDEF with the **ALL** LOB indicator keyword to include both base and LOB table spaces.

Example 4-33 Cross Loader example with a LOB column

```
EXEC SQL
     CREATE DATABASE  PAOLOR3L
  ENDEXEC
  EXEC SQL
     CREATE TABLESPACE DSN8S71B IN PAOLOR3L
            USING STOGROUP  DSN8G710
  ENDEXEC
```

```
EXEC SQL
   CREATE TABLE  PAOLOR3.EXAMP6
     (EMPNO      CHAR(06)   NOT NULL,
      EMP_ROWID  ROWID NOT NULL GENERATED ALWAYS,
      RESUME     CLOB(5K),
      PRIMARY KEY  (EMPNO))
        IN  PAOLOR3L.DSN8S71B
ENDEXEC
EXEC SQL
   CREATE UNIQUE INDEX  PAOLOR3.XEXAMP6
          ON  PAOLOR3.EXAMP6
     (EMPNO  ASC)
          USING  STOGROUP DSN8G710
ENDEXEC
EXEC SQL
      CREATE LOB TABLESPACE  DSN8S71N
                IN  PAOLOR3L
               LOG  NO
ENDEXEC
EXEC SQL
      CREATE AUX TABLE  PAOLOR3.AUX_EMP_RESUME
                IN  PAOLOR3L.DSN8S71N
            STORES  PAOLOR3.EXAMP6
            COLUMN  RESUME
ENDEXEC
EXEC SQL
      CREATE UNIQUE INDEX  PAOLOR3.XAUX_EMP_RESUME
                ON  PAOLOR3.AUX_EMP_RESUME
ENDEXEC
EXEC SQL
  DECLARE C6 CURSOR FOR
  SELECT  EMPNO,
          RESUME
  FROM DSN8710.EMP_PHOTO_RESUME
ENDEXEC
LISTDEF LISTLOB INCLUDE TABLESPACE PAOLOR3L.* ALL
TEMPLATE TSYSUT1  DSN(PAOLOR3.&DB..&TS..SYSUT1) SPACE(10,10) CYL
TEMPLATE TSORTOUT DSN(PAOLOR3.&DB..&TS..SORTOUT) SPACE(10,10) CYL
LOAD DATA
 INCURSOR C6
 INTO TABLE  PAOLOR3.EXAMP6 WORKDDN(TSYSUT1,TSORTOUT)
RUNSTATS TABLESPACE LIST LISTLOB INDEX(ALL)
```

Cross Loader examples with remote source table and local target table

Cross Loader can transparently load data from a remote location connected through DRDA definitions. Appendix B, “DB2 connectivity” on page 307 contains details on setting up the necessary connectivity functions.

In Example 4-34 we use the Cross Loader to transfer data from a **DB2 UDB database on a UNIX server** to DB2 for z/OS. The table in the UDB UNIX database is reached through **DRDA** using its three part name.

Example 4-34 Cross Loader example loading from one remote location

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM SAMPAIX.DB2INST1.DEPARTMENT
  ORDER BY 1 DESC
```

```
ENDEXEC
LOAD DATA
  REPLACE
  INCURSOR C1
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-35 we use the Cross Loader to transfer data from both a DB2 UDB database on a **UNIX server** and from a **Windows NT server** to DB2 for z/OS.

Example 4-35 Cross Loader example loading from two remote locations

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM SAMPAIX.DB2INST1.DEPARTMENT
  ORDER BY 1 DESC
ENDEXEC
EXEC SQL
  DECLARE C2 CURSOR FOR
  SELECT *
  FROM SAMPWIN.DB2INST1.DEPARTMENT
  ORDER BY 1 DESC
ENDEXEC
LOAD DATA
  REPLACE
  INCURSOR C1
  INTO TABLE PAOLOR2.DEPT
LOAD DATA
  RESUME YES
  INCURSOR C2
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-36 we try to use the Cross Loader to transfer the same data as in Example 4-35, but we are using UNION ALL within one EXEC SQL. Referencing object from two different locations is not allowed:

```
   SQLCODE = -512, ERROR:  STATEMENT REFERENCE TO REMOTE OBJECT IS INVALID
```

Example 4-36 Cross Loader example loading from two remote locations and one EXEC SQL

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM SAMPAIX.DB2INST1.DEPARTMENT
  UNION ALL
  SELECT *
  FROM SAMPWIN.DB2INST1.DEPARTMENT
  ORDER BY 1 DESC
ENDEXEC
LOAD DATA
  REPLACE
  INCURSOR C1
  INTO TABLE PAOLOR2.DEPT
```

In Example 4-37 we Load a **partitioned table space** with the Cross Loader. In this case we only use one cursor. The source and target tables are identical.

Example 4-37 Cross Loader example of populating a partitioned table space

```
EXEC SQL
  CREATE TABLESPACE TSSTAFFP
    USING STOGROUP SGLP0002
    PRIQTY 4000 SECQTY 1000
    BUFFERPOOL BP0
    NUMPARTS 4
ENDEXEC
EXEC SQL
  CREATE TABLE PAOLOR2.STAFFP
    (ID               SMALLINT NOT NULL,
     NAME             VARCHAR(9),
     DEPT             SMALLINT,
     JOB              CHAR(5),
     YEARS            SMALLINT,
     SALARY           DECIMAL(7, 2),
     COMM             DECIMAL(7, 2))
  IN TSSTAFFP
ENDEXEC
EXEC SQL
  CREATE UNIQUE INDEX PAOLOR2.XSTAFFP0
    ON PAOLOR2.STAFFP
     (ID              ASC )
    USING STOGROUP SGLP0002
    PRIQTY 800 SECQTY 200
    CLUSTER
     (PART 1 VALUES(00100) ,
      PART 2 VALUES(00200) ,
      PART 3 VALUES(00300) ,
      PART 4 VALUES(00400) )
    BUFFERPOOL BP2
ENDEXEC
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM SAMPAIX.DB2INST1.STAFF
  ORDER BY 1
ENDEXEC
LOAD DATA
  RESUME NO
  INCURSOR C1
  INTO TABLE PAOLOR2.STAFFP
```

In Example 4-38 we Load the same partitioned table space using **partition parallelism**. Therefore, we declare one different cursor per partition, transferring the data belonging to that particular partition.

Example 4-38 Coss Loader example of partition parallelism

```
EXEC SQL
  DECLARE 1A CURSOR FOR
    SELECT *
    FROM SAMPAIX.DB2INST1.STAFF
    WHERE ID <=  100
    ORDER BY 1
ENDEXEC
EXEC SQL
  DECLARE 1B CURSOR FOR
    SELECT *
```

```
    FROM SAMPAIX.DB2INST1.STAFF
    WHERE ID BETWEEN 101 AND 200
    ORDER BY 1
ENDEXEC
EXEC SQL
  DECLARE 1C CURSOR FOR
    SELECT *
    FROM SAMPAIX.DB2INST1.STAFF
    WHERE ID BETWEEN 201 AND 300
    ORDER BY 1
ENDEXEC
EXEC SQL
  DECLARE 1D CURSOR FOR
    SELECT *
    FROM SAMPAIX.DB2INST1.STAFF
    WHERE ID >= 301
    ORDER BY 1
ENDEXEC
LOAD DATA
  REPLACE
  INTO TABLE PAOLOR2.STAFFP PART 1 INCURSOR 1A
  INTO TABLE PAOLOR2.STAFFP PART 2 INCURSOR 1B
  INTO TABLE PAOLOR2.STAFFP PART 3 INCURSOR 1C
  INTO TABLE PAOLOR2.STAFFP PART 4 INCURSOR 1D
```

Example 4-39 verifies that the partitions was loaded in parallel.

Example 4-39 Job report from the parallel loading

```
DSNUGUTC -  LOAD DATA REPLACE
DB2G DSNURWI -  INTO TABLE PAOLOR2.STAFFP PART 1 INCURSOR 1A
DB2G DSNURWI -  INTO TABLE PAOLOR2.STAFFP PART 2 INCURSOR 1B
DB2G DSNURWI -  INTO TABLE PAOLOR2.STAFFP PART 3 INCURSOR 1C
DB2G DSNURWI -  INTO TABLE PAOLOR2.STAFFP PART 4 INCURSOR 1D
DB2G DSNURRST - EXISTING RECORDS DELETED FROM TABLESPACE
  DSNURPPL - PARTITIONS WILL BE LOADED IN PARALLEL, NUMBER OF TASKS = 4
DB2G DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=10 FOR TABLE PAOLOR2.STAFFP PART=1
DB2G DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=10 FOR TABLE PAOLOR2.STAFFP PART=2
DB2G DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=10 FOR TABLE PAOLOR2.STAFFP PART=3
DB2G DSNURWT - (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=5 FOR TABLE PAOLOR2.STAFFP PART=4
  DSNURILD - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=35
  DSNURILD - (RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:15
  DSNUGSOR - SORT PHASE STATISTICS -
             NUMBER OF RECORDS=35
             ELAPSED TIME=00:00:00
DB2G DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=10 FOR INDEX PAOLOR2.XSTAFFP0  PART 1
DB2G DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=10 FOR INDEX PAOLOR2.XSTAFFP0  PART 2
DB2G DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=10 FOR INDEX PAOLOR2.XSTAFFP0  PART 3
DB2G DSNURBXA - BUILD PHASE STATISTICS - NUMBER OF KEYS=5 FOR INDEX PAOLOR2.XSTAFFP0  PART 4
  DSNURBXD - BUILD PHASE STATISTICS - NUMBER OF INDEXES=1
  DSNURBXD - BUILD PHASE COMPLETE, ELAPSED TIME=00:00:00
  DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

Example 4-40 shows loading from a cursor declared on a distributed ASCII table to three different tables on the mainframe:

- PAOLOR2.EMPE is EBCDIC
- PAOLOR2.EMPA is ASCII
- PAOLOR2.EMPU is UNICODE

See Example 4-29 for more details about the tables.

Example 4-40 Cross Loader conversion when loading from distributed to mainframe

```
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT * FROM SAMPAIX8.DB2INST2.EMPLOYEE
  ORDER BY 1 DESC
ENDEXEC
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPE
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPA
LOAD DATA
  INCURSOR C1 REPLACE
  INTO TABLE PAOLOR2.EMPU
```

4.3 Conclusions and recommendations

The Cross Loader function combines the flexibilty of the SQL statement and the performance of the Load utility. The source can either be on the mainframe, in a distributed system or in any system accessible via a *Federated Database*. It eliminates the need for file transfers.

Cross loading is done using the new DB2 V7 option **INCURSOR** in the DB2 Load utility. With this option you tell the Load to read the data from the result set of a cursor instead of reading it from a sequential file. The data is read from the source location with a dynamic SQL statement (enclosed between EXEC SQL and ENDEXEC statements) and loaded in a table at the target location where the the Load utility runs.

The same cursor can be reused multiple times in the utility job step. It can be referenced in different LOAD statements to load the same data in different target tables. It can also be reused in one LOAD statement containing multiple INTO TABLE clauses to LOAD the same data in different target tables (in the same table space.)

If the encoding scheme of the source data in the cursor and the target table differ, DB2 automatically converts the encoding schemes. An example may be conversion from EBCDIC to UNICODE or from ASCII to EBCDIC.

Always sort the input data in clustering sequence before loading. Do this with an ORDER BY clause when using the Cross Loader.

In addition to define a cursor, use EXEC SQL to execute dynamic SQL statements before, between or after utility statements. It can simplify the JCL coding by eliminating dynamic SQL applications like DSNTIAD or DSNTEP2 from the JCL stream and it enables to merge different utility steps, separated by dynamic SQL applications, into one single utility step.

5

Export and Import with DB2 distributed

This chapter provides a description of Export and Import utilities within the DB2 UDB on distributed platforms (UNIX and Windows.)

The following topics will be discussed in this chapter:

- General description of Export utility
- Export examples
- General description of Import utility
- Import examples

Some cross-platform examples are reported in Chapter 11, “Moving data to DB2 Distributed” on page 267, and Chapter 10, “Moving data to DB2 for z/OS” on page 243.

5.1 Export utility overview

The Export utility exports data from a database to an operating system file, or pipe, which can be in one of several external file formats. This operating system file can then be used to move the table data to a different server.

Export utility runs on the client node, hence the output file will be created on the client node as well. Version of the utility being executed depends on the version of DB2 installed on the client machine.

The following information is required when exporting data:

- An SQL SELECT statement specifying the data to be exported.
- The path, name and format of the operating system file that will store the exported data. This format can be IXF, WSF, or DEL.
- When exporting typed tables, you may need to provide the subtable traverse order within the hierarchy. If the IXF format is to be used, the default order is recommended. When specifying the order, recall that the subtables must be traversed in the PRE-ORDER fashion. When exporting typed tables, you cannot provide a SELECT statement directly. Instead, you must specify the target subtable name, and optionally a WHERE clause. The Export utility uses this information, along with the traverse order, to generate and execute the required SELECT statement.

The maximum size of a single LOB value that can be inlined in the export file is 32 KB. If the source table contains larger LOB columns LOB data should be exported into separate a file (or multiple files) by specifying the LOBSINFILE modifier. This is the only way to avoid data truncation.

If you want to use the export data from a partitioned database, you can use **db2batch** to complete the task at each database partition (see 2.3.3, “DB2 db2batch” on page 14, for more information). The SELECT statement must be able to return only the data found locally.

A table can be saved by using the Export utility and specifying the IXF file format. The saved table (including its indexes) can then be recreated using the Import utility.

For further information consult Chapter 1 of *DB2 UDB Data Movement Utilities Guide and Reference Version 8,* SC09-4830*.*

5.2 Using Export utility

This section describes the use of the DB2 UDB Export utility for distributed platforms. We discus how to prepare your database before using the Export, the restrictions on the utility, and some examples on its use.

Preparing for the Export

- You must be connected to (or be able to implicitly connect to) the database from which the data will be exported.
- You must have SYSADM or DBADM authority, or CONTROL or SELECT privilege for each table participating in the export operation.
- Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking Export.
- Note that this utility does not support tables with structured type columns.

5.2.1 Invoking the Export utility

There are three ways you can call the Export utility:

- Command Line Processor (CLP)
- Export Notebook in the Control Center
- Application Programming Interface (API) - sqluexpr

Using the Command Line Processor

From the CLP issue the following command:

```
export to staff.ixf of ixf messages staff.msg select * from userid.staff
```

In this simple example:

- 'staff.ixf' is the output file, its format is specified as PC/IXF
- All messages returned by the utility will be placed in file 'staff.msg'
- Select statement indicates that all source table columns be included in the output

Large object data should be exported as following:

```
export to emp_photo.del of del lobs to lobs/ lobfile emp_photo modified by lobsinfile
messages emp_photo.msg select empno, picture from emp_photo
```

In this example:

- Only two columns (empno and picture) are exported
- Non LOB data is exported to a delimited ASCII file 'emp_photo.del'
- Utility messages are placed in file 'emp_photo.msg'
- As LOBSINFILE modifier is used, LOB data stored in an external file
- LOB data is stored in the lobs directory, and the base name used is emp_photo (if multiple lob files need to be used, a three digit sequence number will be appended to this base name.)

Prior to DB2 UDB V7.2 FixPak 7, Export utility creates a single file for each LOB object (in later releases a lob file contains multiple LOB objects, as limited by the file system file size restrictions.) If a large number of records containing LOB data needs to be exported both 'lobs to' and 'lobfile' can be specified. The utility can export up to 1000 lob objects for each basename specified by the lobfile parameter. Multiple lob paths can be used to circumvent file system space limitations:

```
export to emp_resume.ixf of ixf
lobs to lobs1/, lobs2/
lobfile e_r1.lob, e_r2.lob, e_r3.lob, e_r4.lob, e_r5.lob, e_r6.lob, e_r7.lob, e_r8.lob,
e_r9.lob, e_r10.lob, e_r11.lob, e_r12.lob, e_r13.lob, e_r14.lob, e_r15.lob, e_r16.lob,
e_r17.lob, e_r18.lob, e_r19.lob, e_r20.lob
modified by lobsinfile messages emp_resume.msg select * from emp_resume
```

In this example:

- A maximum of 20000 records can be exported because 20 basenames for LOB files are specified.
- LOB data is exported to lobs1 directory, lobs2 directory is used only if the file system hosting lobs1 fills up.

Using the Export Notebook of the Control Center

Screen shots in this chapter show the Control Center in DB2 UDB V8. If you run a different version of DB2 UDB, graphical interfaces may not be exactly as shown:

1. From the Control Center, expand the object tree until you find the **Tables** folder.

2. Click on the **Tables** folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane). See Figure 5-1 on page 82.
3. Right-click the table you want in the contents pane, and select **Export** from the pop-up menu. The Export notebook opens.
4. Use the **Target** tab to input the target and message file names, select the file type and choose the select statement to be used by the Export utility. DEL file modifiers can be specified by clicking the **Options** button. Figure 5-2 on page 83.
5. Use the **Columns** tab to select and reorder the output columns (this feature is available only for IXF files) and to specify LOB options. Figure 5-3 on page 83.
6. Use the **Schedule** tab to create a scheduled task in the Task Center. Figure 5-4 on page 84.
7. Click the **Show Command** button at any time to display the syntax of the prepared export.

.

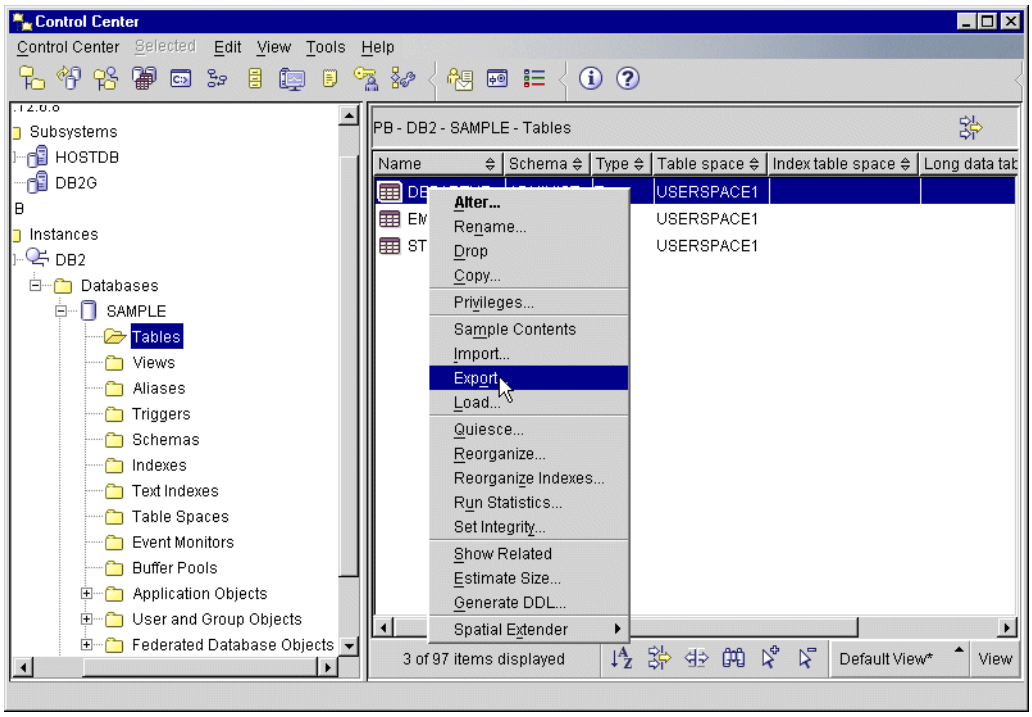

Figure 5-1 The expanded table view in the Control Center

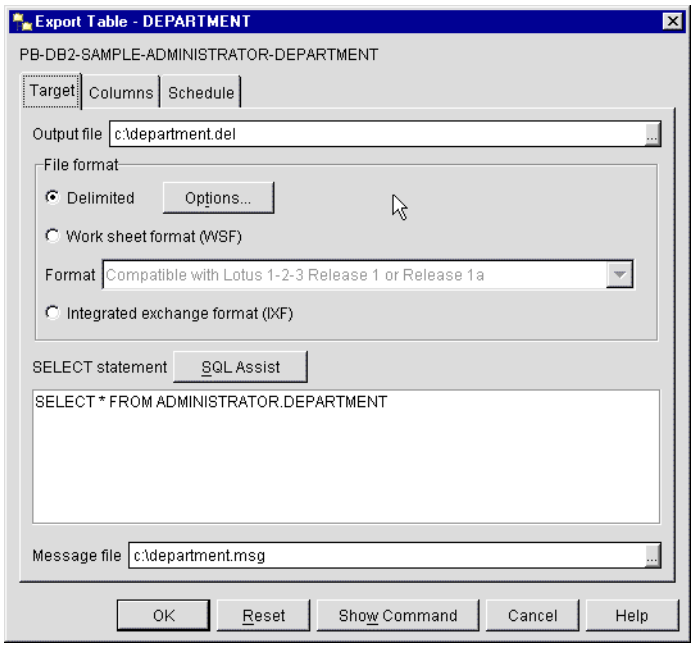

Figure 5-2 The Target tab of the Export Notebook

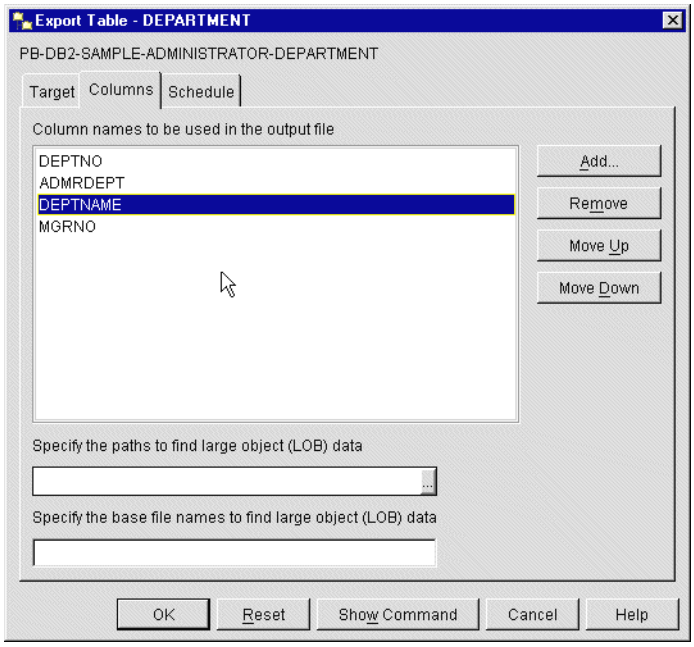

Figure 5-3 The Columns tab of the Export Notebook

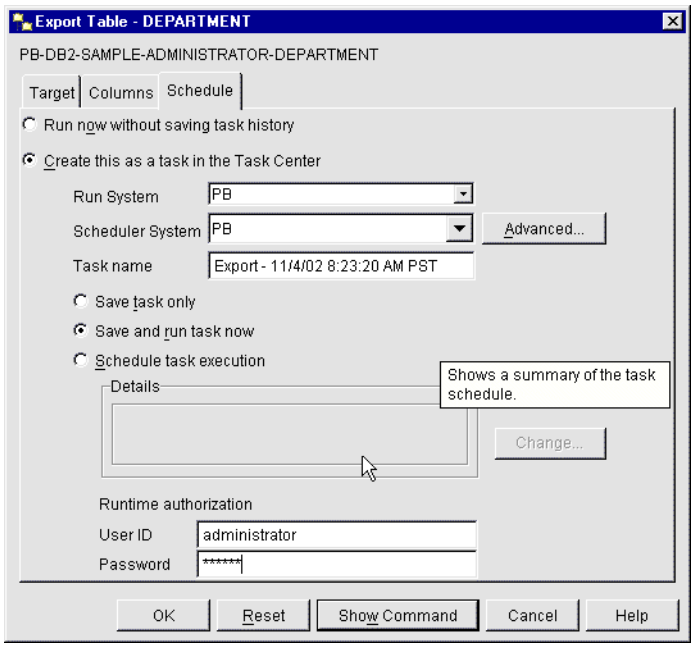

Figure 5-4 The Schedule tab of the Export Notebook

If you need more information on the operation of the Control Center you can view the online Help facility inside the control center.

Export API - sqluexpr

Export utility can be invoked through the Application Programming Interface. A database connection needs to exist before the export is performed (alternatively, implicit connection needs to be enabled.)

Examples of the export API invocation exist in the samples directory, inside the sqllib directory. The files of interest are samples/c/expsamp.sqc, samples/c/impexp.sqc and samples/c/tload.sqc (DB2 UDB V7), and samples/c/tbload.sqc, samples/c/tbmove.sqc and samples/cpp/tbmove.sqC (DB2 UDB V8.)

The parameters for the Export utility API can be found in the *IBM DB2 UDB Administrative API Reference*, SC09-4824.

5.3 Import utility overview

The Import utility uses SQL insert statements to insert data from an input file into a table or updatable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

Import utility runs on the client node, hence the source file has to be located on the client node as well. Version of the utility being executed depends on the version of DB2 installed on the client machine.

When invoking the Import utility you need to specify the following:

- The path, name, and type of the input file. Import utility supports DEL, ASC, PC/IXF, and WSF file formats.

- Name or alias of target table or a view. If the table already exists, data can be appended or replaced. If the target table does not exist, it can be created by the Import utility provided that the source file type is PC/IXF.
- When working with typed tables, you may need to provide the method or order by which to progress through all of the structured types. The order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy is called the *traverse* order. This order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

You can use the Import utility to recreate a table that was saved through the Export utility. The table must have been exported to an IXF file, and the SELECT statement used during the export operation must have met certain conditions (for example, no column names can be used in the SELECT clause; only select * is permitted). When creating a table from an IXF file, not all attributes of the original table are preserved. For example, referential constraints, foreign key definitions, and user-defined data types are not retained.

By default, the Import utility is bound to the database with isolation level RR (repeatable read). If a large number of rows is being imported into a table, the existing lock may escalate to an exclusive lock. If another application working on the same table is holding some row locks, a deadlock will occur if the lock escalates to an exclusive lock. To avoid this, the Import utility requests an exclusive lock on the table at the beginning of its operation. Holding a lock on the table has two implications. First, if there are other applications holding a table lock, or row locks on the Import target table, the Import utility will wait until all of those applications commit or roll back their changes. Second, while Import is running, any other application requesting locks will wait until the Import operation has completed.

In a partitioned database environment, the Import utility can be enabled to use buffered inserts. This reduces the messaging that occurs when data is imported, resulting in better performance; however, since details about a failed buffered insert are not returned, this option should only be enabled if you are not concerned about error reporting.

For further information consult Chapter 2 of *DB2 UDB Data Movement Utilities Guide and Reference Version 8,* SC09-4830.

5.4 Using the Import utility

This section describes the use of the DB2 UDB Import utility for distributed platforms. We discus how to prepare your database before using the Import, the restrictions on the utility, and some examples on its use.

Preparing for the Import

- You must be connected to (or be able to implicitly connect to) the database into which the data will be imported.
- The authority needed to perform Import:
  - To create a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database
  - To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view
  - To append data to an existing table or view, you must have SELECT and INSERT privileges for the table or view.

- Since the utility will issue a COMMIT or a ROLLBACK statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking Import.

Restrictions

- This utility does not support the use of nicknames.
- If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only appended to.
- You cannot perform an Import replace operation into an underlying table of a materialized query table defined in refresh immediate mode.
- You cannot Import data into a system table, a summary table, or a table with a structured type column.
- You cannot Import data into declared temporary tables.
- Views cannot be created through the Import utility.
- Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using SELECT *.)
- Because the Import utility generates its own SQL statements, the maximum statement size of 64KB may, in some cases, be exceeded.
- Import utlity logs all record inserts. If log space is scarce specify a commitcount option to have the utility periodically issue commit statements to empty the logs and release locks.
- Import utility holds an exclusive lock on the target table. If concurrency is an issue, use Load with ALLOW READ ACCESS option to give other applications concurrent (read only) access to the target table.

Note: If an error occurs during an IMPORT REPLACE operation, the original data in the table is lost. If COMMITCOUNT was specified, data already committed will be available immediately after the failure. Retain a copy of the input data to allow the Import operation to be restarted.

5.4.1 Invoking the Import utility

There are three ways you can call the Import utility:

- Command Line Processor (CLP)
- Import Notebook in the Control Center
- Application Programming Interface (API) - sqluimpr

Using the Command Line Processor

To create a copy of the staff table using the Import utility:

```
import from staff.ixf of ixf commitcount 100 messages staff.msg replace_create into
staff2
```

In this example:

- Data is imported from a PC/IXF file staff.ixf, create option is only supported for this file type.
- If the target table staff2 exists its data will be replaced by the records from the input file; if the target table does not exist it will be created using the DDL information contained in the PC/IXF file.

– A commit statement will be executed after every 100 records are imported.

– Utility messages will be placed into file ‘staff.msg’.

To Import data from a positional ASCII file, with LOB data saved in an external file:

```
import from emp_photo.asc of asc lobs from lobs/ modified by lobsinfile reclen= 31
method l (1 6, 8 17, 19 30) null indicators (7, 18, 31) messages emp_photo.msg insert
into emp_photo
```

In this example:

– The target table is truncated before any new data is inserted, note that rows may still be rejected if there are unique index violations.

– Since the input data is ASC a method parameter must specify columns’ positions inside the fixed length records.

– Positions of null indicators are also specified.

– LOB data is read from the lobs directory.

If an unique index is defined on the target table insert_update mode can be used:

```
import from emp_resume.ixf of ixf messages emp_resume.msg insert_update into emp_resume
```

In this example records from the input file will be inserted if there is no primary key match, if a primary key match is found table record will be updated with new data.

To improve performance in partitioned environments:

```
connect to sample
bind db2uimpm.bnd insert buf
import from staff.del of del modified by compound=20 messages staff.msg insert into
staff
```

In this example the Import packages were rebound to use buffered inserts. Compound inserts are used to improve performance, instead of a single statement, a set of 20 insert statements is passed from the client to the server in a single request.

Using the Import Notebook of the Control Center

Screen shots in this chapter show the Control Center in DB2 UDB V8. If you run a different version of DB2 UDB, graphical interfaces may not be exactly as shown:

- From the Control Center, expand the object tree until you find the Tables folder.
- Click the **Tables** folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane.)
- Right-click the table you want in the contents pane, and select **Import** from the pop-up menu. The Import notebook opens.
- Use the **File** tab to input the output and message file names, select the output file type and import mode, and choose ASCII file modifiers (the **Options** button). See Figure 5-5 on page 88.
- Use the **Columns** tab to define the mapping between input data and target columns; choose identity and generated column behavior and set LOB options. See Figure 5-6 on page 88.
- A graphical column mapper is available when ASC file is used. See Figure 5-7 on page 89.

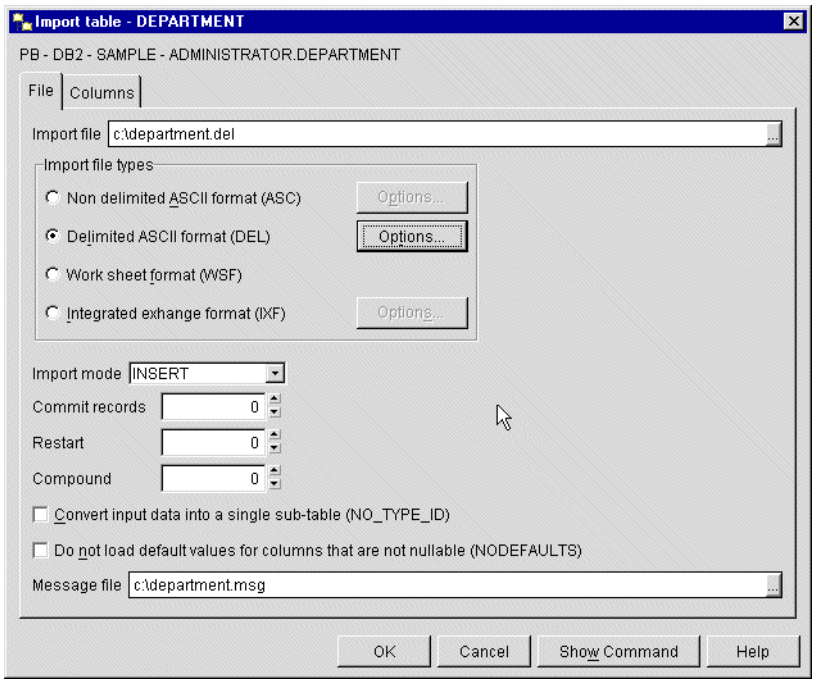

Figure 5-5 The File tab of the Import Notebook

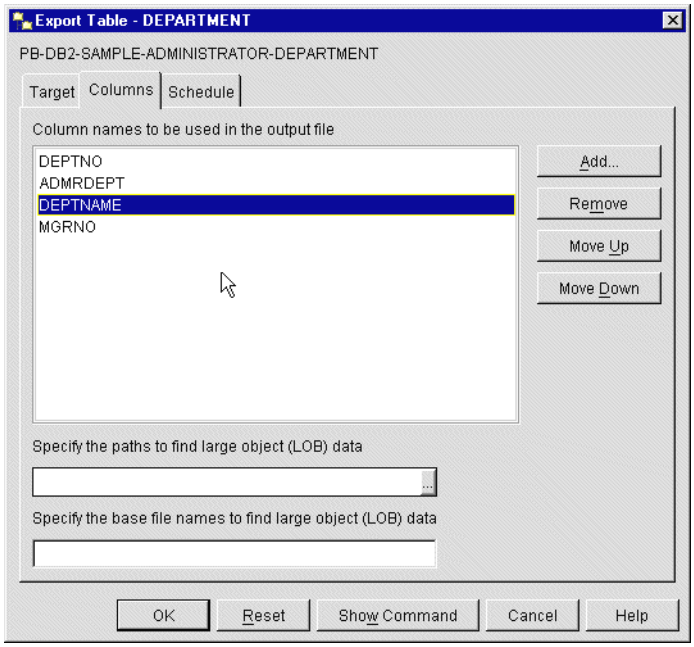

Figure 5-6 The Columns tab of the Import Notebook

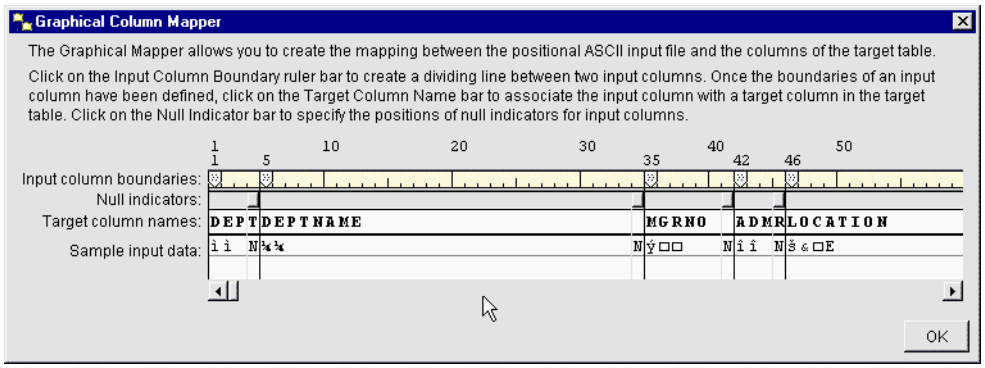

Figure 5-7 The Graphical Column Mapper

If you need more information on the operation of the Control Center you can view the online Help facility inside the control center.

Import API - sqluimpr

Import utility can be invoked through the Application Programming Interface. A database connection needs to exist before the Import is performed (alternatively, implicit connection needs to be enabled.)

Examples of the Import API invocation exist in the samples directory, inside the sqllib directory. The files of interest are samples/c/expsamp.sqc and samples/c/impexp.sqc (DB2 UDB V7), and samples/c/dtformat.sqc, samples/c/tbmove.sqc and samples/cpp/tbmove.sqC (DB2 UDB V8.)

The parameters for the Import utility API can be found in the *IBM DB2 UDB Administrative API Reference*, SC09-4824.

6

Load with DB2 Distributed

In this chapter, we discuss the DB2 UDB Load utility for distributed platforms, and the new features introduced in DB2 UDB V8. The advantages and disadvantages of using this utility are also analyzed.

These are the topics contained in this chapter:

- Load utility overview
- AutoLoader utility
- Using the Load utility
- Comparing Load and Import
- Load and Import functional comparison
- When to use Load or Import utilities

6.1 Load utility overview

The Load utility moves data from files, named pipes, or cursors into a DB2 table. Input data sources can reside either on one of the database nodes, or on a remotely connected client. The table being loaded must exist. If the target table already contains data, you can replace or append to the existing data.

The Load utility is capable of efficiently moving large quantities of data into empty tables, or into tables that already contain data. It can handle most data types, including large objects (LOBs), user-defined types (UDTs), and DATALINKs. It is much faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. It does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes.)

The Load utility runs on the server. The version of the utility being executed depends on the version of DB2 installed on the server machine. There are restrictions on the Load options available if Load is invoked through a down level client.

There are several processes that happen internally to ensure the integrity of the data and the efficient performance of the database. On each partition where the target table resides, the work performed by the Load utility can be categorized into different phases. In this section, we will discuss these per-partition phases.

6.1.1 Per-partition Load operation

There are four distinct phases of the Load operation for each partition where the target table resides. Of the four phases, only the *Load phase* is guaranteed to take place. Execution of any of the other three phases depends on the characteristics of the table, the data being loaded, and the Load options specified. All of the phases are executed automatically by the Load utility once the Load request is made.

Phase 1: Load phase

During the Load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. Consistency points are established at intervals specified through the SAVECOUNT parameter in the LOAD command. For each consistency point reached, a Load message is generated, indicating how many input rows were successfully loaded up to that time. For DATALINK columns defined with FILE LINK CONTROL, link operations are performed for non-NULL column values, and violations are recorded so that the rows that caused them can be removed in the delete phase.

Messages about rejected rows are written to the Load message file. Following the completion of the Load process, review these messages and correct problems appropriately.

If a failure occurs during the Load phase, you can restart the Load operation; the RESTART option automatically restarts the Load operation from the last successful consistency point. The TERMINATE option rolls back the failed Load operation. Note that REPLACE option requires a table truncation before new data is inserted. This truncation is irreversible.

Phase 2: Build phase

If index key processing was performed during the Load phase, the build phase will follow. During this phase, indexes are built based on the index keys collected during the Load phase. The index keys are sorted during the build phase, and index statistics are collected (if the STATISTICS YES with INDEXES option was specified). The statistics are similar to those collected through the RUNSTATS command. Additionally, while unique indexes are being

built, unique key violations are recorded. Rows that cause these violations will be removed in the delete phase.

If a failure occurs during the build phase, the RESTART option automatically restarts the Load operation at the appropriate point.

Phase 3: Delete phase

Rows that caused violations (whether through unique key violations or violations through DATALINK columns defined with FILE LINK CONTROL) will be deleted in the delete phase. Unique key violations are placed into the exception table, if one was specified. Following the completion of the Load process, review these messages, resolve any problems, and insert corrected rows into the table.

If a failure occurs during the delete phase, the RESTART option automatically restarts the Load operation at the appropriate point.

Note: Each deletion event is logged. If you have a large number of records that violate the uniqueness condition, the log could fill up during the delete phase. Load utility also logs splits done on index object pages.

Phase 4: Index copy phase

If a Load was invoked with the ALLOW READ ACCESS option in conjunction with the USE option, index data was temporarily written to a table space specified by the USE option. The index copy phase will take place at the end of Load to copy the index data from that table space to the proper table space where the index was defined.

If a failure occurs during the index copy phase, the RESTART option automatically restarts the Load operation at the appropriate point.

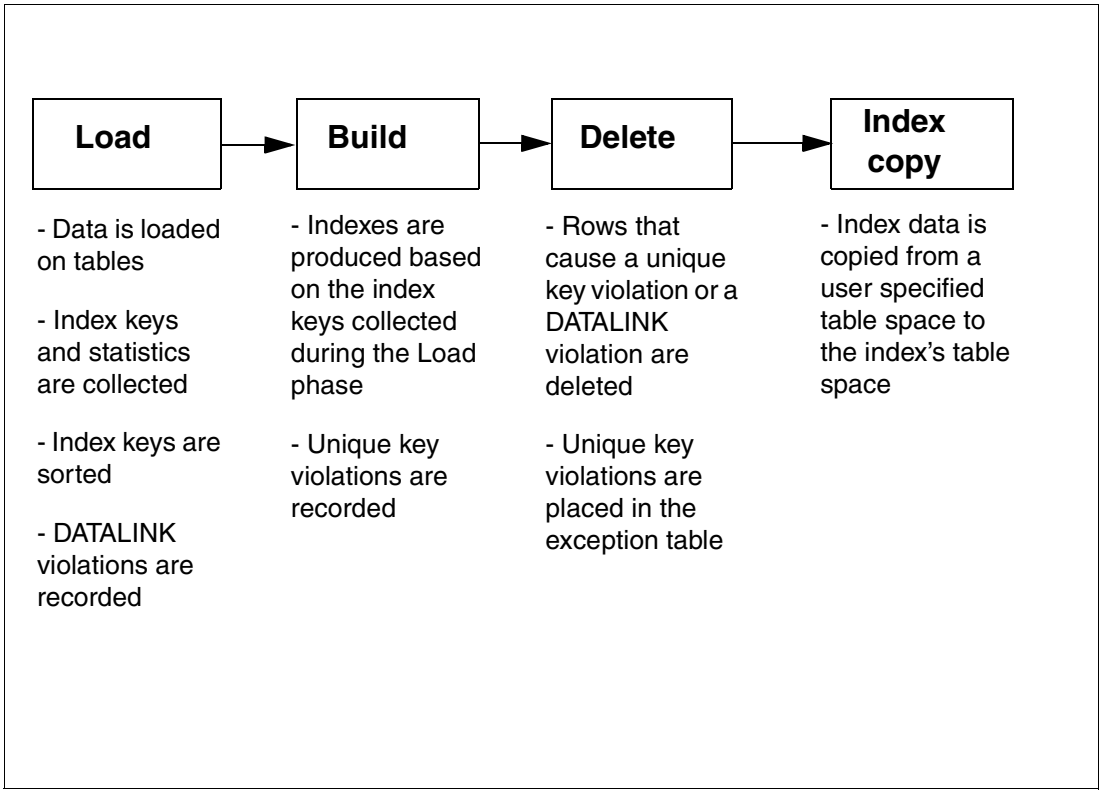

Figure 6-1 The four phases of the per-partition Load process

6.1.2 Load Recovery

If the target table's database is retaining logs for recovery purposes (in other words, its *logretain* configuration parameter set to *on*), it is possible to specify whether a Load should keep a copy of the changes made. This is done to enable roll forward recovery of the database. This option is not supported if forward log recovery is disabled for the database; that is, if the database configuration parameters *logretain* and *userexit* are disabled. If no copy is made, and forward log recovery is enabled, the tablespace is left in a *backup pending* state at the completion of the Load operation.

When *logretain* is *on*, loads are run with one of the following recovery options:

- COPY YES
  - A copy of the loaded data is generated as the Load takes place (COPY YES.) Subsequent roll forward processing will process the copy and update the table accordingly.
- COPY NO
  - No copy is taken (which will speed up processing), however, the tablespaces where the loaded table is defined are placed in *backup pending*. This means that the only allowable updates to those tablespaces will be through running the Load utility. Performing reads to tables in those table spaces is still permitted, however. The backup pending state persists until a backup of those tablespaces (or the entire database) is taken.
- NONRECOVERABLE
  - No copy is taken, however, the tablespaces where the loaded table is defined are *not* placed in backup pending. This is meant as a convenience so that access to table spaces will not be restricted after loads are performed. However, there is risk associated with this convenience. If roll forward processing encounters a

nonrecoverable Load to a table, it will place that table in a *drop pending* state, so that after the roll forward completes, the only operation allowed to that table will be to drop it.

6.2 AutoLoader utility

If the target table resides in a partitioned database, each row of data needs to be partitioned to the correct partition before being loaded. Starting with DB2 UDB V7, the AutoLoader executable *db2atld* is used to load data into a table in a partitioned database environment. It can be used in one of the following modes:

SPLIT_AND_LOAD

Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

SPLIT_ONLY

Data is partitioned (perhaps in parallel) and the output is written to files, one file for each partition that the table is defined on. Each output file contains header information that includes the partition number that it is associated to, and the table's partitioning map. This is done so that db2atld can later be run in the LOAD_ONLY mode to load the output files. No loading takes place.

LOAD_ONLY

Output files generated by a previous run of db2atld in the SPLIT_ONLY mode are loaded simultaneously on the corresponding database partitions. No partitioning takes place.

ANALYZE

An optimal partitioning map with even distribution across all database partitions is generated, according to the provided data. This map can be used to redefine a table's partitioning scheme to prevent a skewed distribution.

In its most common mode, SPLIT_AND_LOAD, db2atld partitions input data into as many output sockets as there are database partitions in which the target table is defined. While doing this, it performs a Load operation (see 6.1.1, "Per-partition Load operation" on page 92 for details) concurrently on each of these partitions, where each Load reads partitioned data from its designated output socket. A key feature of db2atld is that it uses direct TCP/IP communication using sockets for all data transfer required during both the partitioning and loading processes. It is also capable of running multiple processes to parallelize the partitioning of data, thereby significantly improving performance.

db2atld supports delimited (DEL) and positional (ASC) ASCII files. It is not possible to use this utility to load PC/IXF files.

Input parameters for db2atld are set in the AutoLoader configuration file. An example is shown in Section 6.4, "Using the Load utility" on page 99.

For further information on the AutoLoader executable please refer to Chapter 4 of the *DB2 UDB Data Movement Utilities Guide and Reference Version 7,* SC09-4830.

6.3 New features in DB2 distributed V8

In this section we provide a brief introduction to the new feature available with DB2 UDB V8. Detailed descriptions can be found in Chapters 3 and 4 of the *DB2 UDB Data Movement Utilities Guide and Reference Version 8,* SC09-4830.

6.3.1 Increased table space access during Load

Starting with Version 8, the Load utility no longer quiesces the tablespaces that the table to be loaded are defined on. It also no longer places those table spaces in Load pending for the duration of the Load. This means that all other tables in those table spaces are fully accessible (both reading and writing). The only table space state that can be set as a result of a Load is the backup pending state (see “COPY NO” on page 94 for details). Access to the table being loaded is disallowed unless the newly introduced “ALLOW READ ACCESS” option of Load is specified (see 6.3.2, “Load with read access” on page 96 for details.)

6.3.2 Load with read access

In most cases, the Load utility uses table level locking to restrict access to tables. The level of locking depends on whether or not the Load operation allows read access. A Load operation in ALLOW NO ACCESS mode will use an exclusive lock (Z-lock) on the table for the duration of the Load. An Load operation in ALLOW READ ACCESS mode acquires and maintains a share lock (S-lock) for the duration of the Load operation, and upgrades the lock to an exclusive lock (Z-lock) when data is being committed.

Before a Load operation in ALLOW READ ACCESS mode begins, the Load utility will wait for all applications that began before the Load operation to release locks on the target table. Since locks are not persistent, they are supplemented by table states that will remain even if a Load operation is aborted. These states can be checked by using the LOAD QUERY command. By using the LOCK WITH FORCE option, the Load utility will force applications holding conflicting locks off the table that it is trying to load into.

At the beginning of a Load operation in ALLOW READ ACCESS mode, the Load utility acquires a share lock (S-lock) on the table. It holds this lock until the data is being committed. The share lock allows applications with compatible locks to access the table during the Load operation. For example, applications that use read only queries will be able to access the table, while applications that try to insert data into the table will be denied. When the Load utility acquires the share lock on the table, it will wait for all applications that hold locks on the table prior to the start of the Load operation to release them, even if they have compatible locks. Since the Load utility upgrades the share lock to an exclusive (Z-lock) when the data is being committed, there may be some delay in commit time while the Load utility waits for applications with conflicting locks to finish.

6.3.3 Load into partitioned databases

Starting with DB2 UDB V8, all the functionality of the db2atld executable (autoloader) has been incorporated into the Load utility. Some of the new features of the Load utility in a partitioned database environment (also known as partitioned load) include:

- Partitioned load can now be performed from the Command Line Processor (CLP) or invoked through an calling the db2Load() API from an application program.
- Partitioning of the input data is possible even if the target table’s partitioning key includes generated columns or column default values.
- The *CLIENT* option of Load is supported for a partitioned load.
- Cross Loader functionality is supported for a partitioned load (see 6.3.4, “Cross Loader option” on page 98 for details.)

The db2atld executable is no longer needed, since all of the modes and options supported by the AutoLoader can now be used with the Load utility to achieve the same results. To preserve backward compatibility, however, a db2atld executable is included in DB2 UDB V8. It

reads the AutoLoader configuration file and invokes the new db2Load() API to perform a partitioned load. *This new implementation is transparent to the user.*

It is recommended that the *messages* option be specified for a partitioned load. If it is not specified, Load will only display the final sqlcode reported by each of the loading and partitioning processes, and other warnings and errors encountered during processing will not be made available to the user.

Available modes for Version 8 partitioned database Load

The supported modes of the Load utility in a partitioned database environment are as follows:

- PARTITION_AND_LOAD

  Equivalent to the SPLIT_AND_LOAD mode for db2atld (see “SPLIT_AND_LOAD” on page 95 for more details.)

- PARTITION_ONLY

  Equivalent to the SPLIT_ONLY mode for db2atld (see “SPLIT_ONLY” on page 95 for more details.)

- LOAD_ONLY

  Same as the LOAD_ONLY mode for db2atld (see “LOAD_ONLY” on page 95 for more details.)

- LOAD_ONLY_VERIFY_PART

  This is a newly introduced mode. The LOAD_ONLY_VERIFY_PART mode is similar to the LOAD_ONLY mode, in that one file is loaded for each partition that the target table is defined on. However, each file must not contain the header information that is normally present in an output file generated by the PARTITION_ONLY mode. For each partition where a Load operation is taking place, extra work will be performed during the Load phase (see , “Phase 1: Load phase” on page 92 for more details) to verify that each row read actually belongs to the partition it is being read on. Rows that are found to be on the wrong partition are rejected.

- ANALYZE

  Same as the ANALYZE mode for db2atld (see “ANALYZE” on page 95 for more details.)

Loading IXF files in a Version 8 partitioned database environment

Partitioning is not supported for PC/IXF files. However, starting Version 8, it is possible to load PC/IXF files into a partitioned database environment using the LOAD_ONLY_VERIFY_PART mode. The following is an example of how to accomplish this:

Suppose a user wishes to load a PC/IXF datafile “myfile.ixf” into a table “mytable” that is defined on three partitions -- 0, 1, and 2. A directory “/mypath” exists and is writable by the user on all three partitions. The user must:

- On each partition, make a complete copy of myfile.ixf, whose name is myfile.ixf.*nnn*, where *nnn* is the partition number which must be 3 digits (padded with zeroes if necessary). The copy should be placed in a directory that exists on all partitions. Since /mypath is one such directory, the copy can be placed there.
- Issue a Load command that is similar to the following:

```
Load from myfile.ixf of ixf replace into mytable partitioned db config mode
LOAD_ONLY_VERIFY_PART part_file_location /mypath
```

Note that each partition will be attempting to load a complete copy of the PC/IXF datafile. This means that many rows will be rejected in each partition. The number of rows read will be *n*r* , where *n* is the number of partitions where the target table is defined on, and *r* is the number of

rows in the PC/IXF datafile. The number of rows rejected will be at least r*(n-1), which represents the total number of rows that were found to be on the wrong partition.

6.3.4 Cross Loader option

The Cross Loader, also known as “Load From Cursor”, allows data resulting from an SQL query to be directly loaded into a target table without the need to create an intermediate exported file. The SQL query used by Cross Loader operates under a connection to the database in which the target table resides. This does not mean, however, that if a cross loading from a source table to a target table is being performed, the source and target table must reside in the same database. Through Federated Databases a nickname for the source table can be created in the target table’s database, so that the SQL query used by the crossload (into the target table) can perform a select out of the nickname. For details on setting up a federated database, see Appendix A, “Defining a Federated Database” on page 299.

To execute a crossload through CLP, a cursor must first be declared against an SQL query. Once this is done, simply execute the Load specifying the cursoriness as the source, and CURSOR as the source type. Please refer to 6.4.1, “Invoking the Load utility” on page 100 for an example.

The same result can be achieved through calling the db2Load API in an embedded SQL application. For an example, see Example 6-1 on page 108.

6.3.5 Generated column support

The Load utility can be used to load data into a table containing an identity column. If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the Load utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The Load utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values will not be reported.

The Load utility can be used to load data into a table containing (non-identity) generated columns. The column values will be generated by this utility. If no generated column-related file type modifiers are used, the Load utility works according to the following rules:

- Values will be created for generated columns when the corresponding row of the data file is missing a value for the column or a NULL value is supplied. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).
- If a NULL value is created for a generated column that is not nullable, the entire row of data will be rejected (SQL0407N). This could occur if, for example, a non-nullable generated column is defined as the sum of two table columns that include NULL values in the data file.

6.3.6 Multi-dimensional clustering support

Starting with DB2 UDB V8, table data can be clustered along multiple dimensions. Loading into clustered tables is fully supported. The following restrictions apply to multi-dimensional clustering (MDC) tables:

- The SAVECOUNT option of the LOAD command is not supported.
- The TOTALFREESPACE file type modifier is not supported since these tables manage their own free space.

When using the LOAD command with MDC, violations of unique constraints will be handled as follows:

- If the table included a unique key prior to the load operation and duplicate records are loaded into the table, the original record will remain and the new records will be deleted during the delete phase.
- If the table did not include a unique key prior to the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key will be loaded and the others will be deleted during the delete phase.

> **Note:** There is no explicit technique for determining which record will be loaded and which will be deleted.

Performance considerations

To improve the performance of the Load utility when loading MDC tables, the UTIL_HEAP_SZ database configuration parameter should be set to a value that is 10-15% higher than usual. This will reduce disk I/O during the clustering of data that is performed during the load phase. When the DATA BUFFER option of LOAD command is specified, its value should also be increased by 10-15%. If the LOAD command is being used to load several MDC tables concurrently, the UTIL_HEAP_SZ configuration parameter should be increased accordingly.

MDC load operations will always have a build phase since all MDC tables have block indexes.

During the load phase, extra logging for the maintenance of the block map will be performed. There are approximately two extra log records per extent allocated. To ensure good performance, the LOGBUFSZ database configuration parameter should be set to a value that takes this into account.

A system temporary table with an index is used to load data into MDC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

6.4 Using the Load utility

This section describes the use of the DB2 UDB Load utility for distributed platforms. We discuss how to prepare your database before using the Load, the restrictions on the utility, and some examples on its use.

Preparing for the Load

- You must be able to connect to the target database.
- The authority needed to perform the Load is any of the following:

  - sysadm
  - dbadm
  - Load authority on the database and insert privilege on the table. Delete privilege on the table would needed if the REPLACE mode, TERMINATE mode or REPLACE mode is invoked in the Load utility.
- There should be no locks existing on the target table.
- All transactions or unit of work must be complete. If there are some unfinished transactions a COMMIT or a ROLLBACK must first be performed.
- Source data should be sorted in the way that you intend it to be loaded on the target database. This is an optional step to perform if you intend to load your data in a particular sequence.
- If your table is not using Multi-Dimensional Clustering(MDC) and clustering is required, you should sort the data with respect to the clustering index before loading it.

Restrictions

The Load utility cannot perform the following operations:

- Load data into a declared temporary table
- Load data into nicknames
- Load data into hierarchical tables
- Load data into typed tables, or tables with structured type columns
- Create tables before loading into them
- Load data into a database accessed through DB2 Connect or a server level that is not supported
- Activate triggers on newly loaded rows, business rules associated with triggers are not enforced by the Load utility.

Note: If an error occurs during a LOAD REPLACE operation, the original data in the table is lost. Retain a copy of the input data to allow the Load operation to be restarted.

6.4.1 Invoking the Load utility

There are three ways you can call the Load utility:

- Command Line Processor (CLP)
- Load Wizard in the Control Center
- Application Programming Interface (API) - db2Load() in DB2 UDB V8 or sqluload() in DB2 UDB V7

Using the Command Line Processor

For the following Load command:

```
load from inputtab.ixf of ixf messages inputerr.msgs insert into userid.staff copy yes use tsm data buffer 4000
```

In this example:

- ‘inputtab.ixf’ is the source PC/IXF file.
- ‘inputerr.msgs’ is the file where warning and error messages are placed. While loading into a single partition database, if the messages option is not specified, messages will be sent to the standard output, which can affect performance.

– A copy of the inserted data is stored in Tivoli Storage Manager. A copy target can only be specified for databases with either the *LOGRETAIN* or *USEREXIT* database configuration parameters set.
– 4,000 4 KB pages of buffer space from the utility heap are to be used during the Load operation.

Another example of using Load in the CLP is:

```
load from inputtab.ixf of ixf messages inputerr.msgs tempfiles path /u/myuser replace into
staff
```

In this example:

– Table data is being replaced, as specified by the ‘replace into’ option.
– The TEMPFILES PATH parameter is used to specify /u/myuser as the server path into which temporary files will be written.

Note: These examples use relative path names for the Load input file. Relative path names are only allowed on Load requests from a locally connected client. The use of fully qualified path names is recommended.

The following example performs an “online” Load, allowing other applications read access to table data. The example is specific to DB2 UDB V8:

```
load from inputtab.ixf of ixf messages inputerr.msgs
insert into staff indexing mode rebuild allow read access use tmpspace
```

In this example:

– Load is online, as ‘allow read access’ is specified.
– Table indices are temporarily rebuilt in a table space named tmpspace before the index table space data is copied over to its defined location.

The following example loads a delimited ASCII file into a table residing in a partitioned database. Therefore, the example is specific to DB2 UDB V8:

```
load from inputtab.del of del messages inputerr.msgs
replace into staff partitioned db config mode partition_and_load
partitioning_dbpartnums(0,1) output_dbpartnums(1,2,3)
```

In this example:

– Options specific to partitioned database environments are specified after the ‘partitioned db config’ parameter.
– As the *PARTITION_AND_LOAD* mode is used, data is both partitioned and loaded simultaneously.
– Data partitioning is performed on partitions 0 and 1.
– Load operations are being performed on partitions 1, 2 and 3.

The following example, which is specific to DB2 UDB V8, performs a cross load, loading the contents from one table into another.

```
connect to mydb
DECLARE mycurs CURSOR FOR SELECT TWO,ONE,THREE FROM abc.table1
LOAD FROM mycurs OF cursor messages messages.txt INSERT INTO abc.table2
connect reset
```

In this example:

– Tables abc.table1 and abc.table2 in database mydb exist with the following column definitions:

```
abc.table1:
ONE INT
TWO CHAR(10)
THREE DATE

abc.table2:
ONE VARCHAR(20)
TWO INT
THREE DATE
```

– All the data contained inside abc.table1 was loaded into abc.table2, saving Load messages to the file messages.txt:

The complete syntax for the Load command can be interactively checked from the online Command Reference or by issuing ‘db2 ? load’ command from the command window.

Using the Load Notebook of the Control Center

Screen shots in this chapter show the Control Center in DB2 UDB V8. If you run a different version of DB2 UDB, graphical interfaces may not be exactly as shown.

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane.)
3. Click the right mouse button on the table you want in the contents pane, and select Load from the pop-up menu. The Load Wizard opens.
4. Use the Type page to choose load mode and read access options. See Figure 6-2.

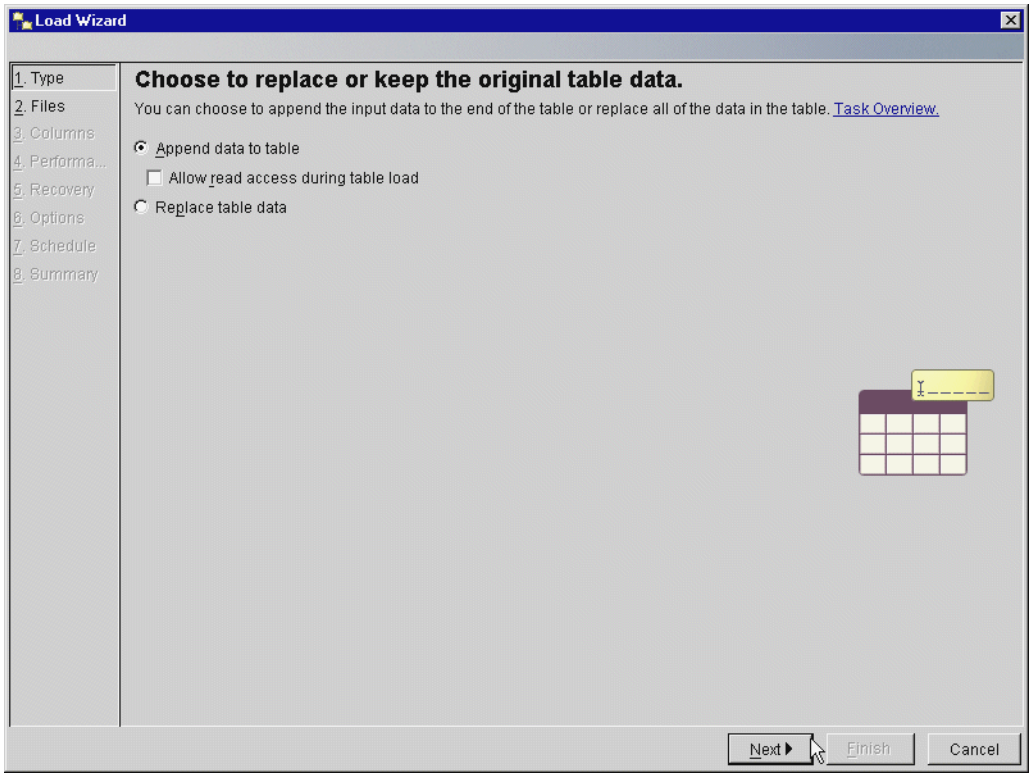

Figure 6-2 Type page of the Load Wizard

5. Use the File page to select file type and location and to choose source file and message file names. See Figure 6-3.

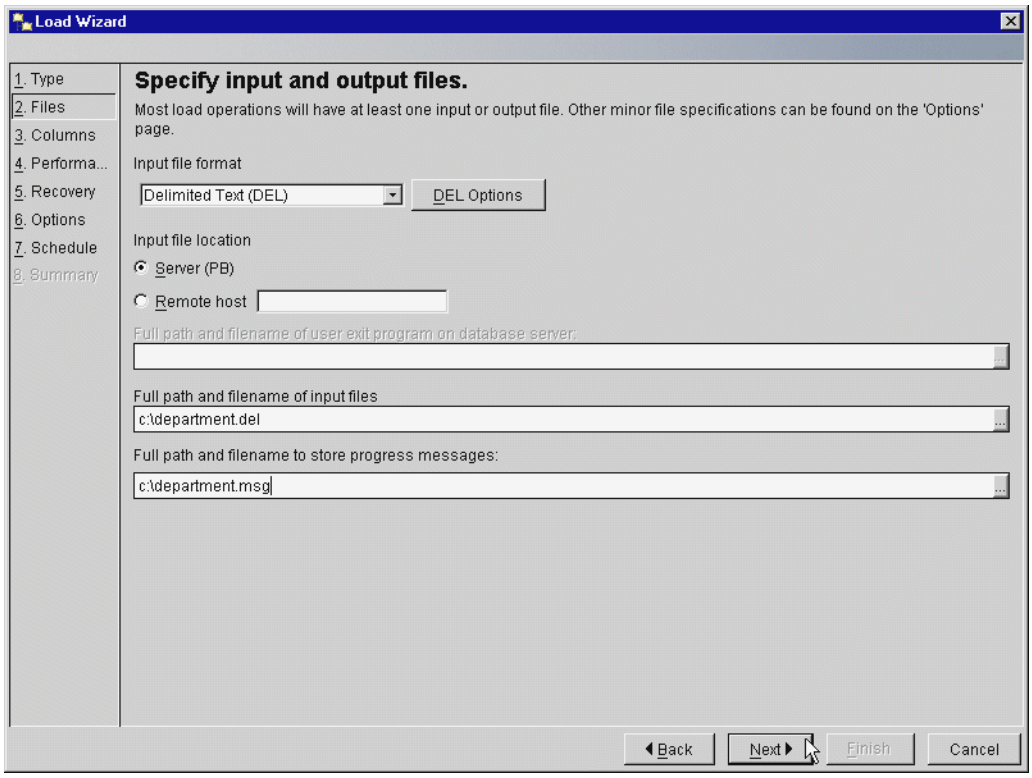

Figure 6-3 Files page of the Load Wizard

6. Use the Columns page to define the mapping between input data and target columns, choose identity and generated column behavior and set LOB options. See Figure 6-4.

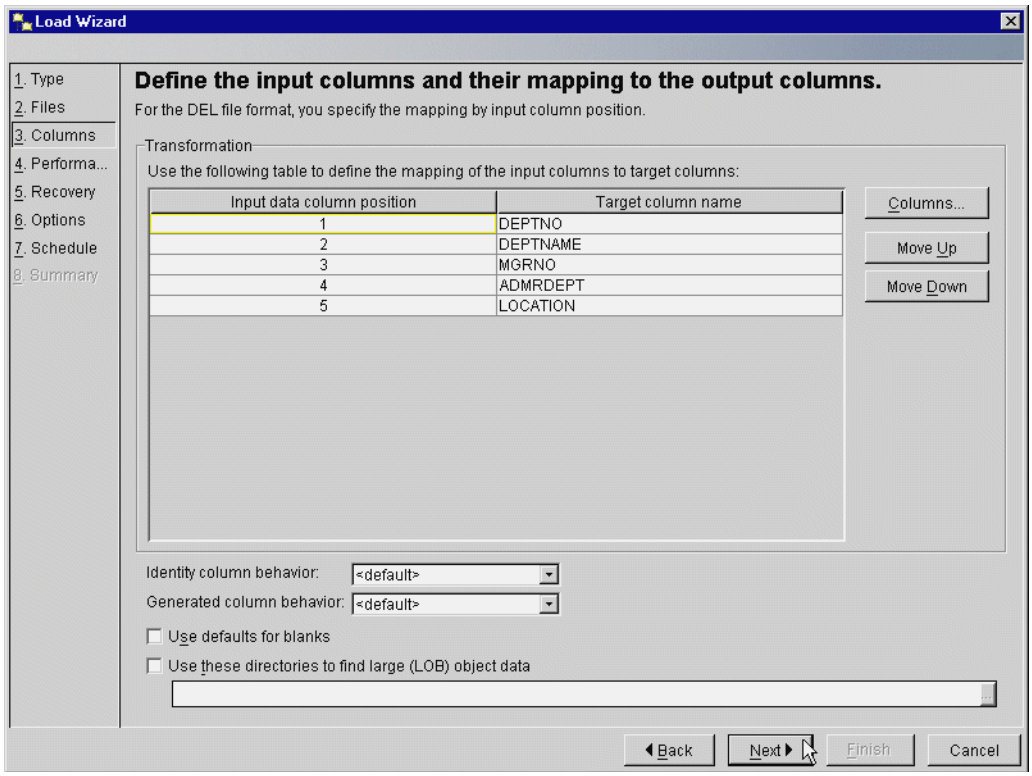

Figure 6-4 Columns page of the Load Wizard

7. As in the Import Notebook, A graphical column mapper is available when ASC file is used. Figure 5-7 on page 89.
8. Use the Performance page to choose minimal checking (FASTPARSE modifier), check pending option and force option, to set the index build mode and statistics mode, and to select free page options. See Figure 6-5.

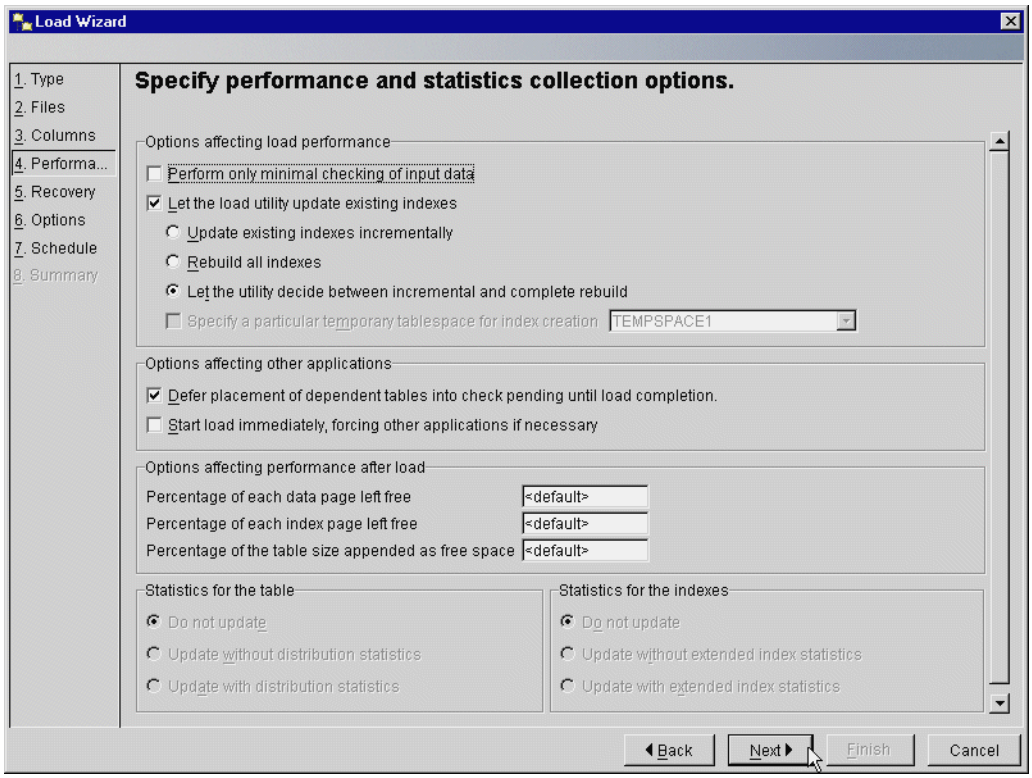

Figure 6-5 Performance page of the Load Wizard

9. Use the Recovery page to set the crash recovery options and the forward recovery options (Load copy options). See Figure 6-6.

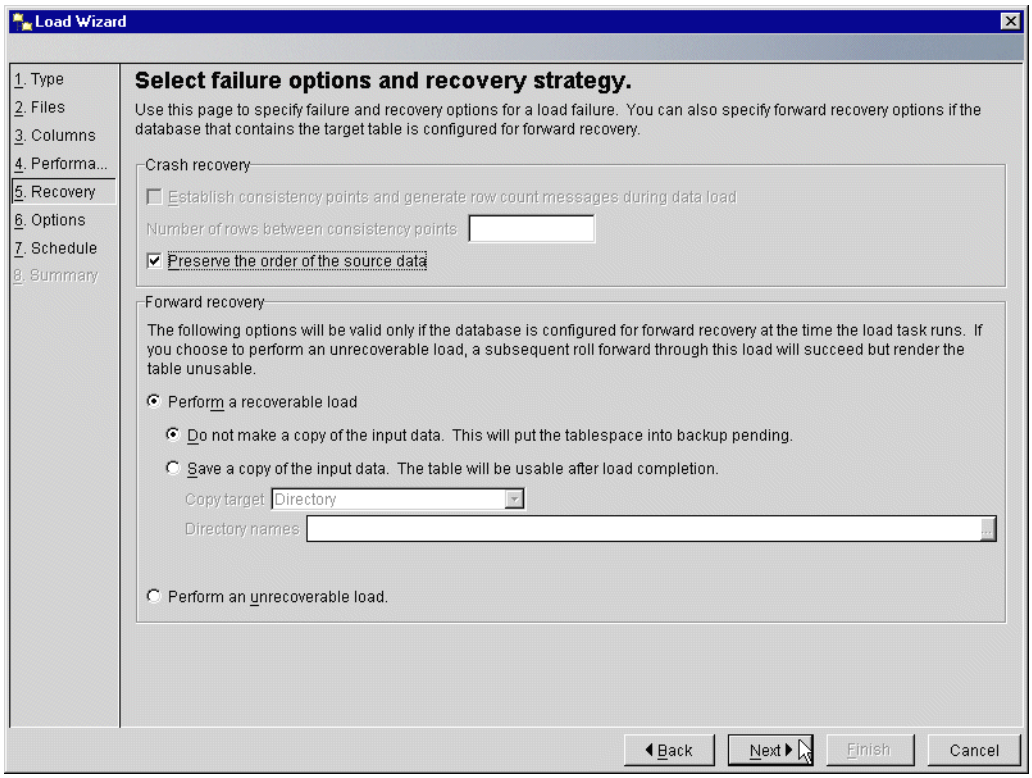

Figure 6-6 Recovery page of the Load Wizard

10. Use the Options page to set more advanced Load options. Hint section gives the explanation of the selected option. See Figure 6-7.
11. Use the Schedule page to create a scheduled task in the Task Center. See Figure 6-8.
12. Before executing the command, Review the selected Load task shown on the Summary page. Figure 6-9.

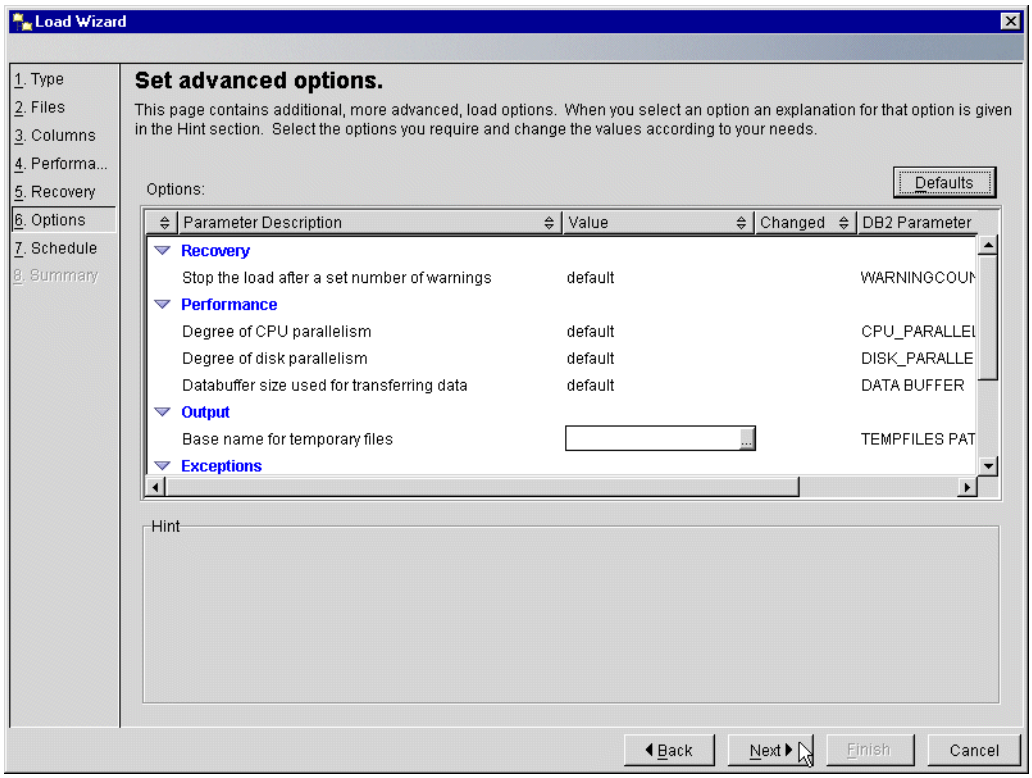

Figure 6-7 Options page of the Load Wizard

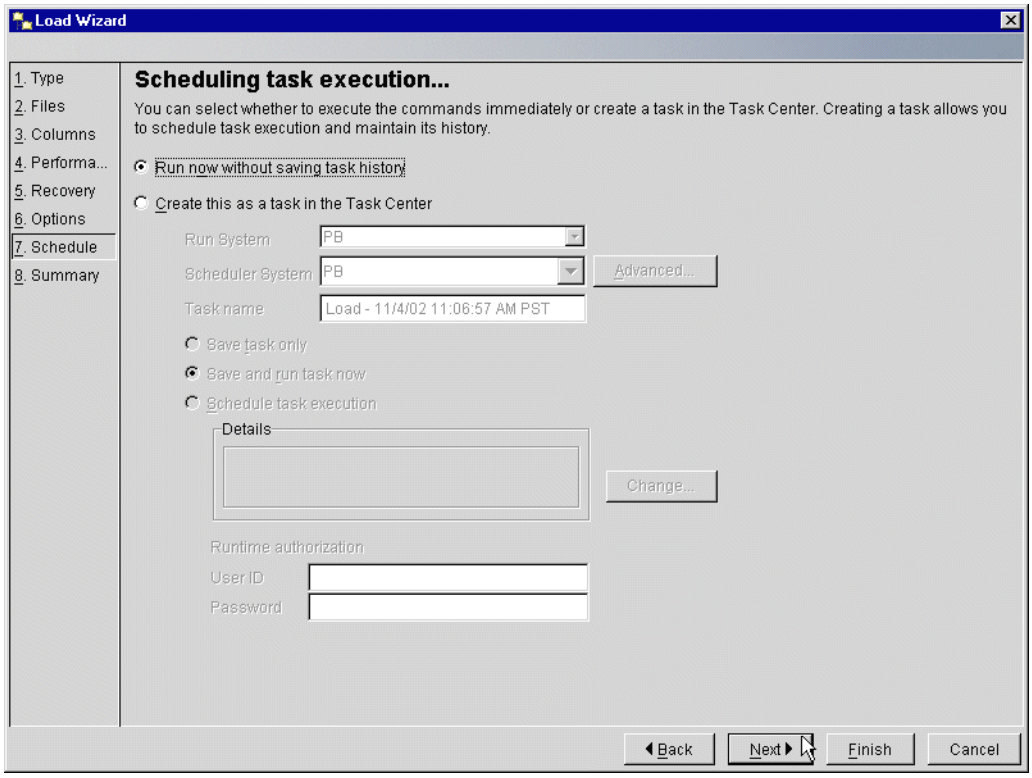

Figure 6-8 Schedule page of the Load Wizard

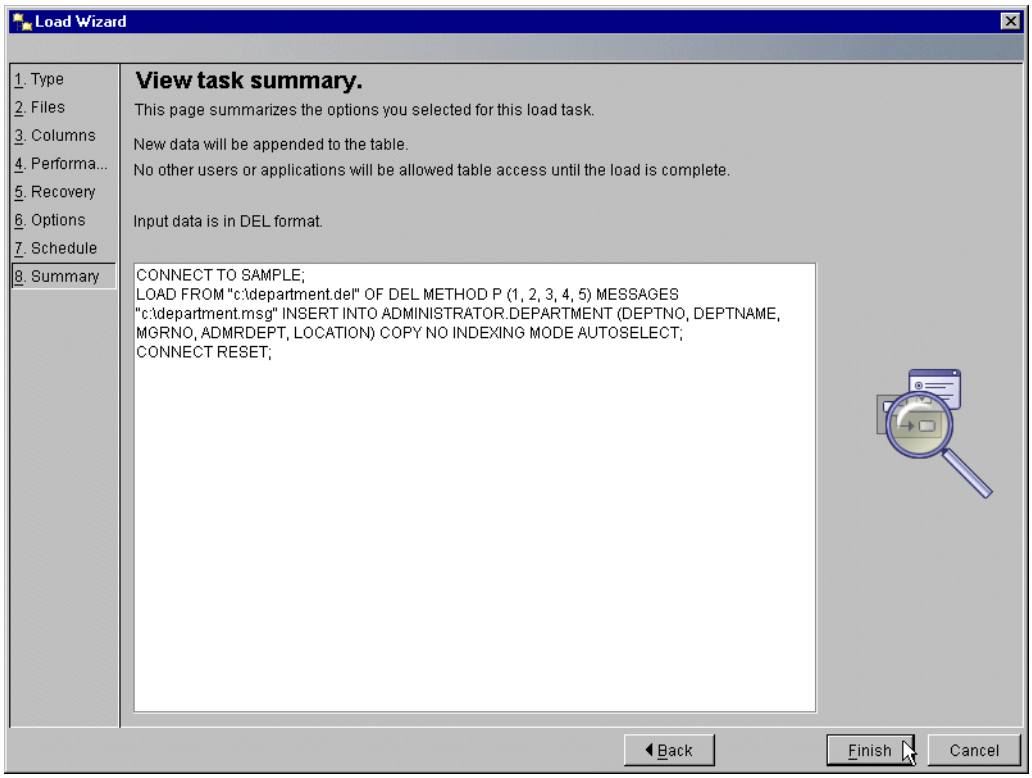

Figure 6-9 Summary page of the Load Wizard

If you need more information on the operation of the Control Center you can view the online Help facility inside the control center.

Load API

The Load utility can be invoked through the Application Programming Interface. A database connection needs to exist before the Load is performed.

As of DB2 UDB V8, the recommended Load API is db2Load(). For previous versions, sqluload() is used.

Examples of the Load API invocation exist in the samples directory, inside the sqllib directory. The files of interest are samples/c/tload.sqc and samples/cobol/tload.sqb (DB2 UDB V7), and samples/c/tbload.sqc, samples/c/tbmove.sqc and samples/c/dtformat.sqc (DB2 UDB V8.)

Example 6-1 demonstrates how the db2Load() API can be used to perform a cross load.

Example 6-1 Embedded SQL — Example of Cross Loader usage

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <db2ApiDf.h>
#include <sqlutil.h>

int loadAndDisplayResults
(
  db2LoadStruct *pLoadStruct,
  struct sqlca *pSqlca
)
{
```

```
  int rc = 0;

  rc = db2Load(db2Version810, pLoadStruct, pSqlca);

  printf("rc from load : %d\n", rc);
  printf("Sqlcode from load : %d\n", pSqlca->sqlcode);
  if (pSqlca->sqlcode == 0)
  {
    db2LoadOut *pLoadOut = pLoadStruct->poLoadInfoOut;
    int iRowsRead = 0;
    int iRowsSkipped = 0;
    int iRowsLoaded = 0;
    int iRowsRejected = 0;
    int iRowsDeleted = 0;
    int iRowsCommitted = 0;

    printf("\n");
    iRowsRead = pLoadOut->oRowsRead;
    printf("Number of rows read      : %d\n", iRowsRead);
    iRowsSkipped = pLoadOut->oRowsSkipped;
    printf("Number of rows skipped   : %d\n", iRowsSkipped);
    iRowsLoaded = pLoadOut->oRowsLoaded;
    printf("Number of rows loaded    : %d\n", iRowsLoaded);
    iRowsRejected = pLoadOut->oRowsRejected;
    printf("Number of rows rejected  : %d\n", iRowsRejected);
    iRowsDeleted = pLoadOut->oRowsDeleted;
    printf("Number of rows deleted   : %d\n", iRowsDeleted);
    iRowsCommitted = pLoadOut->oRowsCommitted;
    printf("Number of rows committed : %d\n", iRowsCommitted);
  }
  else
  {
    char sqlcaString[1024];
    sqlaintp(sqlcaString, 1024, 70, pSqlca);
    printf("%s", sqlcaString);
  }

exit:

  return rc;
}

int main(int argc, char *argv[])
{
  int rc = 0;
  char *pBuffer = NULL;

  struct sqlchar *pActionString = NULL;
  short CallerAction;
  db2LoadIn loadIn;
  db2LoadOut loadOut;
  db2LoadStruct loadStruct;
  char * pLocalMsgFileName;
  sqlu_media_list * pCopyTargetList = NULL;
  sqlu_media_list CopyTargetList;
  sqlu_media_entry MediaEntry;
  sqlu_media_list DataFileList;
  sqlu_statement_entry StatementEntry;
  struct sqldcol DataDescriptor;
  char tempChar[256];
```

```
  char xloadQuery[256];

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;

  char sqlStmt[1024];

EXEC SQL END DECLARE SECTION;

  memset(&CopyTargetList, 0, sizeof(CopyTargetList));
  memset(&MediaEntry, 0, sizeof(MediaEntry));
  memset(&DataFileList, 0, sizeof(DataFileList));
  memset(&StatementEntry, 0, sizeof(StatementEntry));
  memset(&DataDescriptor, 0, sizeof(DataDescriptor));

  memset(&loadIn, 0, sizeof(loadIn));
  memset(&loadOut, 0, sizeof(loadOut));
  memset(&loadStruct, 0, sizeof(loadStruct));

  loadStruct.piSourceList = &DataFileList;
  loadStruct.piLobPathList = NULL;
  loadStruct.piDataDescriptor = &DataDescriptor;
  loadStruct.piFileTypeMod = NULL;
  loadStruct.piTempFilesPath = NULL;
  loadStruct.piVendorSortWorkPaths = NULL;
  loadStruct.piCopyTargetList = NULL;
  loadStruct.piNullIndicators = NULL;
  loadStruct.piLoadInfoIn = &loadIn;
  loadStruct.poLoadInfoOut = &loadOut;

  pLocalMsgFileName = "messages.txt";
  loadStruct.piLocalMsgFileName = pLocalMsgFileName;

  EXEC SQL CONNECT TO mydb;
  printf("[%s] : sqlcode %d\n", "connect to mydb", sqlca.sqlcode);
  printf("\n");

  DataFileList.media_type = SQLU_SQL_STMT;
  DataFileList.sessions   = 1;
  DataFileList.target.pStatement = &StatementEntry;
  DataFileList.target.pStatement->pEntry = xloadQuery;
  DataDescriptor.dcolmeth = SQL_METH_D;
  sprintf(tempChar, "INSERT INTO abc.table2");

  pActionString = (sqlchar *)malloc(strlen(tempChar) +
                                    sizeof(struct sqlchar));

  if (pActionString == NULL)
  {
    printf("Error allocating action string!\n");
    rc = -1;
    goto exit;
  }

  strncpy(pActionString->data, tempChar, strlen(tempChar));
  pActionString->length = strlen((char *)tempChar);
  CallerAction = SQLU_INITIAL;

  loadIn.iRestartphase = ' ';
```

```
  loadIn.iNonrecoverable = SQLU_NON_RECOVERABLE_LOAD;
  loadIn.iStatsOpt = SQLU_STATS_NONE;
  loadStruct.piActionString = pActionString;
  loadStruct.piFileType = SQL_CURSOR;
  loadStruct.iCallerAction = CallerAction;

  sprintf(xloadQuery, "SELECT TWO,ONE,THREE FROM abc.table1");
  DataFileList.target.pStatement->length = strlen(xloadQuery);

  printf("
================================================================================\n");
  printf("    CROSSLOAD STARTING.\n");
  printf("
================================================================================\n");

  loadAndDisplayResults(&loadStruct, &sqlca);

  printf("
================================================================================\n");
  printf("    CROSSLOAD FINISHED.\n");
  printf("
================================================================================\n");

  EXEC SQL CONNECT RESET;
  printf("\n");
  printf("[%s] : sqlcode %d\n", "connect reset", sqlca.sqlcode);

  if (pActionString != NULL)
  {
    free(pActionString);
  }

exit:

  return rc;
}
```

The parameters for the Load utility APIs can be found in the appropriate versions of the *IBM DB2 UDB Administrative API Reference,* SC09-4824.

Sort capability

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a Load is attempted. The Load utility builds indexes based on existing definitions. Exception tables can be used as a repository for unique key violations.

Referential Integrity

The Load utility does not enforce referential integrity, perform constraints checking, or update summary tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in a *check pending* state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in a check pending state. Issue the SET INTEGRITY statement to take the tables out of check pending state. Load operations cannot be carried out on replicated summary tables. For clustering indexes, the data should be sorted on the clustering index prior to loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

6.5 Comparing Load and Import

In this section we compare Load to Import in terms of performance and functionality.

6.5.1 LOAD and Import performance comparison

In this section we present performance comparisons between Load and Import utilities. In the analysis we used tables from the *sample* database (created by the *db2sampl* utility.) Measurements were done on DB2 UDB V7 and V8 for UNIX and Windows platforms.

Utilities were executed in replace mode, with messages parameter specified. Tables from the sample database were populated with randomly generated data.

These results should not be used to compare performance of DB2 UDB V8 to V7 because tests were ran on different physical setups. By repeating some measurement we estimate a 5% error on the ratios of execution times in non-partitioned environments. Because of necessary network traffic and more complex processing model, this uncertainty is larger in partitioned environments. Our estimate here is 10%.

Non-partitioned sample database on Windows platform was used for the analysis summarized in Table 6-1. Target table (staff) does not contain any LOB or LONG columns. No indices are defined on the table. All reported times are in seconds.

Table 6-1 Comparison of Import and Load execution times on Windows platform

DB2	Rows	Format	Import (s)	Load (s)	Ratio
V7	1 M	ASC	620.5	59.6	10.4
V7	1 M	DEL	568.5	51.5	11.0
V7	1 M	IXF	526.9	48.4	10.9
V8	1 M	ASC	113.5	12.7	9.0
V8	1 M	DEL	95.5	12.1	7.9
V8	1 M	IXF	90.4	11.2	8.0

Load utility outperformed import by roughly a factor of 10. Differences between different formats (PC/IXF, delimited ASCII, fixed length positional ASCII) are on the 10% level.

Partitioned and non-partitioned sample databases on UNIX platform were used for the analysis summarized in Table 6-2. Target table (emp_photo) does contain a LOB column. Two dimensional primary key is defined on the table. All reported times are in seconds.

Table 6-2 Comparison of Import and Load execution times on UNIX platform

DB2	Partitions	Rows	Format	Import (s)	Load (s)	Ratio
V7	1	100 K	ASC	200.1	10.2	19.5
V7	1	2 M	ASC	-		-
V7	1	100 K	DEL	198.3	10.0	19.9
V7	1	100 K	DEL, LOBSINFILE	234.5	15.7	14.9
V8	1	100 K	ASC	104.2	18.6	5.6
V8	1	2 M	ASC	-		-

DB2	Partitions	Rows	Format	Import (s)	Load (s)	Ratio
V8	1	100 K	DEL	109.6	18.7	5.9
V8	1	100 K	DEL, LOBSINFILE	159.0	25.5	6.2
V8	4 MLN	100 K	ASC	719.2	60.3	11.9
V8	4 MLN	100 K	DEL	712.6	59.1	12.1
V8	4 MLN	100 K	DEL, LOBSINFILE	827.0	64.5	12.8

Load outperforms Import in all scenarios. The ratio of import execution time to Load execution time in non-partitioned DB2 UDB V8 (a factor of 6) is considerably smaller than in partitioned DB2 UDB V8 and in non-partitioned DB2 UDB V7 (a factor of 12 and 18 respectively.)

Apparent degradation in Load performance in V8 is caused by increased one time cost of starting a Load. For a larger Load (2 million rows into the emp_photo table) execution times measured with V7 and V8 are comparable. Since emp_photo table has a unique index Load goes through the delete phase to remove duplicate rows, which is why execution time does not scale linearly with the number of rows. Consistency of V7 and V8 Load performance was further confirmed by loading 50000000 rows into the projects table of the sample database. The difference between V7 (354 seconds) and V8 (355 seconds) execution times was well within the expected errors.

In a partitioned database environment an additional cost is incurred because data records need to be hashed and send to the right partition. This overhead is more considerable for the Import utility (a factor of 6) than for the Load utility (a factor of 3.)

6.5.2 Load and Import functional comparison

In Table 6-3 we provide a summarized comparison of functions.

Table 6-3 Summary of important differences between the DB2 Load and Import utilities.

Import utility	Load utility
Slow when moving large amounts of data.	Faster than the import utility when moving large amounts of data, because the Load utility writes formatted pages directly into the database.
Limited exploitation of intra-partition parallelism.	Exploitation of intra-partition parallelism. Typically, this requires symmetric multiprocessor (SMP) machines.
No FASTPARSE support.	FASTPARSE support, providing reduced data checking of user-supplied data.
No CODEPAGE support (unless DRDA is utilized.)	CODEPAGE support, converting character data (and numeric data specified in characters) from this code page to the database code page during the Load operation.
Supports hierarchical data.	Does not support hierarchical data.
Creation of tables, hierarchies, and indexes supported with PC/IXF format.	Tables and indexes must exist.
No support for importing into materialized query tables.	Support for loading into materialized query tables.
WSF format is supported.	WSF format is not supported.

No BINARYNUMERICS support.	BINARYNUMERICS support.
No PACKEDDECIMAL support.	PACKEDDECIMAL support.
No ZONEDDECIMAL support.	ZONEDDECIMAL support.
Cannot override columns defined as GENERATED ALWAYS.	Can override GENERATED ALWAYS columns, by using the GENERATEDIGNORE and IDENTITYIGNORE file type modifiers.
Supports import into tables and views.	Supports loading into tables only.
All rows are logged.	Minimal logging is performed.
Trigger support.	No trigger support.
If an import operation is interrupted, and a *commitcount* was specified, the table is usable and will contain the rows that were loaded up to the last COMMIT. The user can resume the import operation, or accept the table as is.	If a Load operation is interrupted, and a *savecount* was specified, the table remains in Load pending state and cannot be used until the Load operation is restarted, a Load terminate operation is invoked, a Load replace operation is invoked, or until the table space is restored from a backup image created some time before the attempted Load operation.
Temporary space required is approximately equivalent to the size of the largest index plus 10%. This space is obtained from the temporary table spaces within the database.	Temporary space required is approximately equivalent to the sum of the size of all indexes defined on the table, and can be as much as twice this size. This space is obtained from temporary space within the database.
All constraints are validated during an import operation.	The Load utility checks for uniqueness and computes generated column values, but all other constraints must be checked using SET INTEGRITY.
The key values are inserted into the index one at a time during an import operation.	The key values are sorted and the index is built after the data has been loaded.
If updated statistics are required, the runstats utility must be run after an import operation.	Statistics can be gathered during the Load operation if all the data in the table is being replaced.
You can import into a host database through DB2 Connect.	You cannot load into a host database.
Input data must reside on the same machine that the import utility is invoked.	Input data can reside on the server, or on the remotely connected client from which the Load utility is invoked.
A backup image is not required. Because the import utility uses SQL inserts, DB2 logs the activity, and no backups are required to recover these operations in case of failure.	A backup image can be created during the Load operation.

6.5.3 When to use Load or Import utilities

This section summarizes strengths and weaknesses of the two utilities used for inserting data from external sources into the database. These results should help your make an informed decision so as to which utility to use in a given scenario.

Use Load when:

- Performance is critical. As shown, Load outperforms Import in all scenarios, sometimes by as much as a factor of 10.

Use Import when:

- Target table resides on a DB2 UDB V7 server and contains generated columns. Load's generated column support in DB2 UDB V7 is limited. For further details consult the appropriate *DB2 UDB Data Movement Utilities Guide and Reference Version 8,* SC09-4830.
- Target table needs to be created from the input file (applicable to PC/IXF files only.)
- Target database is accessed through DB2 Connect, or it resides on a down-level server that is not supported by client's version of the Load utility.
- Target table is a hierarchical table or a typed table.
- Logging of every inserted record is necessary.

Part 3

High Performance Unload

This part is dedicated to the IBM High Performance Unload tool, which is available as separate products for the host and distributed platforms.

We will describe these products in the following chapters:

- IBM DB2 High Performance Unload for z/OS
- IBM DB2 High Performance Unload for Multiplatforms

7

IBM DB2 High Performance Unload for z/OS

In this chapter we discuss the DB2 High Performance Unload for z/OS (HPU for z/OS). We evaluate its functions, the operating instructions, explore the advantages and disadvantages of using the tool. We are using DB2 UDB for z/OS Version 7 and the HPU Version 2.1.

This chapter is structured in:

- An overview of the HPU tool
- The prerequisites of HPU
- Installation and customization
- Data formats used
- Using HPU in batch job
- Using HPU in interactive mode

7.1 An overview of HPU for z/OS

IBM HPU is a DB2 tool for unloading DB2 tables. It can unload DB2 data from a tablespace or an image copy. And it can unload a DB2 table in different file formats.

The HPU provides efficient use of CPU processing power by working down to the VSAM level instead of using SQL and going through DB2. The utility directly accesses the VSAM clusters that DB2 uses to store its tables spaces. This direct use of VSAM takes maximum advantage of VSAM's buffering capability, which lets an entire cylinder be read with a single I/O.

Some other advantages of HPU is its ability to perform parallel unloads of different tables under one tablespace by using multiple SQL statements. It can also unload from image copies, hence lessening the interference with the DB2 production database. It can do selective unload of rows and columns. With the use of HPU user exit, it can inspect, modify, or discard DB2 rows.

7.1.1 Applicability of HPU

Database shops nowadays have ever increasing demand for efficiency. The amount of data that needs to be backed-up increases exponentially and the available time to perform it is constantly bound by the continually shrinking batch window. High availability of data demands high performance in the software and hardware used in database shops.

Reading large amount of data in a sequential manner substantially increases the amount of time needed to complete the unload task. Hence, making this task fit in the 10 to 12 hour batch window that you usually have is becoming harder as time goes by. Efficiency becomes a primary criterion in selecting tools and utilities to be used in the database shop.

HPU helps to increase productivity and task efficiency by performing the data unload in parallel manner. Reading the different tables at the same time and performing several unloads concurrently using an image copy significantly increase the amount of work that can be done in a given time.

The HPU provides an extremely fast way to sequentially read and share a DB2 tablespace among multiple tasks. This is needed to avoid the potential problem in the DB2 buffer pool management when multiple programs compete for the same data. It avoids writing over buffers that may be serving several unloads and potential channel conflicts. By optimizing sequential reads of the table space, HPU will reduce both the elapsed and CPU time needed to execute the unloads.

HPU provides different options for the output file format.

The HPU output format can be:

- VARIABLE, which lets you quickly create variable-length records
- DELIMITED, which lets you quickly create a delimited file that can be exported to another platform
- DSNTIAUL compatible, for movement within mainframe databases
- USER, which lets you specify virtually any type of conversion so that your output appears as you want it

7.1.2 Strong points of HPU

There are the things HPU does to make it stand up to its name. To maximize performance, HPU uses buffering and synchronization techniques:

Buffering When reading data rows, HPU directly accesses the VSAM clusters containing the table space.This direct use of VSAM takes maximum advantage of VSAM's buffering capability, which lets an entire cylinder be read with a single I/O.

Synchronization HPU permits the parallel execution of several unloads that are accessing the same table space; it does this by synchronizing the unloads. Each row extracted from a table space is offered successively to each of the unloads executed under HPU control, so that a single read satisfies several requests.

7.2 Installing HPU for z/OS

This section contains information on how to install HPU in a z/OS environment.

7.2.1 Installation requirements

In a z/OS environment there are two systems involved in the installation of a program: the driving system, which is the system that contains the jobs that will be used to install the program; and the target system, which is the system where the program will be installed.

In most cases, you can use the same system to install the HPU. However, there are cases where it is necessary to use a different driving system and target system. One reason for using a separate installation system is if you want one program having different versions to run in parallel with each other. This is done when you want the older and more stable version to run in the production LPAR (logical partition) and the new and untested version to run in the test LPAR. Another case where it is necessary to use a separate installation system is when the program you are installing shares a program library or load modules with other products running in your system. And you want the installation not to disrupt the programs running in the production LPAR.

Driving system requirements

The software required to install HPU are any of the following:

- OS/390 SMP/E Version 2 Release 8 or higher (includes SMP/E) - 5647-A01
- z/OS Version 1 Release 1 or higher (includes SMP/E) - 5694-A01
- IBM SMP/E for z/OS and OS/390 Version 3 Release 1 or higher - 5655-G44

The hardware requirement is any mainframe hardware (such as machine type 9672, 2066, 2064) that can run the software requirement.

Target system requirements

Mandatory requisites (PREREQ/REQ)

The mandatory requisites are modules needed by the program for it to be *installed* and to ***f****unction properly* in the system where it is installed. These are known as the REQs or PREREQs. The mandatory requisite for HPU is:

- DB2 for OS/390 Version 5 (minimum).

Functional requisites (IFREQ)

Functional requisites are products not required during the installation of the program, but it would be needed *during run-time* for a specific function of this product to work properly. They are known as the IFREQs.

- HPU has no functional prerequisites.

Toleration/coexistence requisites (COREQ)

Toleration/coexistence requisites are products needed by the program so that it can coexist with other programs in a shared environment. Such environments are parallel sysplex, shared DASD, or a system that reuse DASD at different time intervals. These requisites are known as COREQs.

- HPU has no toleration/coexistence requisites.

Negative requisites

Negative or incompatibility requisites are products that cannot exist on the same system with the one being installed. A given product is *mutually exclusive* with its negative requisite.

- HPU has no negative requisites.

Storage requirements

HPU requires:

- 1470 blocks for the target system
- 1470 blocks for the distribution system

Notes: IBM recommends use of system determined block sizes for efficient disk utilization for all non-RECFM U data sets. For RECFM U data sets, IBM recommends a block size of 32760, which is the most efficient from a performance and DASD utilization perspective.

All target and distribution libraries listed have the following attributes:

- The default name of the data set may be changed.
- The default block size of the data set may be changed.
- The data set may be merged with another data set that has equivalent characteristics.
- The data set may be either a PDS or a PDSE.

All target libraries listed have the following attributes:

- The data set may be SMS managed.
- It is not required for the data set to be SMS managed.
- It is not required for the data set to reside on the IPL volume.
- The values in the Member Type column are not necessarily the actual SMP/E element types identified in the SMPMCS.

All target libraries listed which contain load modules have the following attributes:

- The data set may be in the LPA.
- It is not required for the data set to be in the LPA.
- The data set may be in the LNKLST.
- It is not required for the data set to be APF authorized.

Important: Some FMIDs are deleted by the installation.

There are FMIDs that could be deleted when you install the HPU. You should check the ++VER part of the SMPMCS. If you find some FMIDs that you do not want to be deleted you should install the HPU in a different SMP/E target and distribution zone.

7.2.2 Step-by-step installation procedures

The HPU Version 2.1 is installed in the z/OS or OS/390 using the System Modification Program / Extended (SMP/E). It is installed using the usual RECEIVE, APPLY, ACCEPT commands. The SMP/E dialogs may used to do the entire SMP/E installation part.

Step 1: Set SMP/E CSI sub-entries

IBM recommends these values in the CSI sub-entries. If the value is lower than these it will result in failure in the installation steps. See table Table 7-1 for the values recommended. DSSPACE is a subentry in the GLOBAL options entry. PEMAX is a subentry of the GENERAL entry in the GLOBAL options entry. Refer to the SMP/E manuals for instructions on updating the global zone.

Table 7-1 SMP/E CSI sub-entry values

SUB-ENTRY	Value	Comment
DSSPACE	(200,200,500)	3390 DASD tracks
PEMAX	SMP/E Default	IBM recommends using the SMP/E default for PEMAX.

Step 2: Modify and submit JCL to download installation jobs

You can download the installation jobs by submitting the JCL in Example 7-1. You can copy the jobs from tape or from disk. You need to modify the TAPEIN DD or the FILEIN DD statement depending on the installation program vehicle you use.

Example 7-1 JCL to unload the sample jobs to install HPU

```
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=
//*******************************************************************************
//* If you wish to create a new global zone do not run the
//* RCVPDO job supplied with CBPDO..RCVPDO assumes that
//* you will be installing into an existing global zone.
//* Make the ///TAPEIN DD statement below active if you install
//* from a CBPDO tape by uncommenting the DD statement below.
//*******************************************************************************
//TAPEIN DD DSN=IBM.HINZ21.F2,UNIT=tunit,
//          VOL==SER=volser,LABEL=(X,SL),
//          DISP==(OLD,KEEP)
//*******************************************************************************
//* Make the ///TAPEIN DD statement below active if you install
//* from a product tape received outside the CBPDO process
//* (using the optional SMP/E RECEIVE job) by uncommenting
//* the DD statement below.
//*******************************************************************************
//*TAPEIN DD DSN=IBM.HINZ21.F2,UNIT=tunit,
//*          VOL==SER=INZ21,LABEL=(3,SL),
//*          DISP==(OLD,KEEP)
//*******************************************************************************
```

```
//* Make the ///FILEIN DD statement below active for
//* downloaded DASD files.
//*********************************************************************
//FILEIN DD DSN=IBM.HINZ21.F2,UNIT=SYSALLDA,DISP=SHR,
//            VOL=SER= filevol
//OUT    DD  DSNAME=jcl-library-name,
//            DISP=(NEW,CATLG,DELETE),
//            VOL=SER= dasdvol,UNIT=SYSALLDA,
//            SPACE=(88,(15,5,5))
//SYSUT3  DD  UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSIN   DD
COPY INDD= indd, OUTDD=OUT
SELECT MEMBER=(INZALA,INZALB,INZALLOC,INZDDDEF,INZRECEV)
SELECT MEMBER=(INZAPPLY,INZACCEP)
//
```

In the download JCL, **tunit** is the unit value matching the product tape, **volser** is the volume serial which is described in the CBPDO documentation, **X** is the tape file number where the data set name is on the CBPDO tape (refer to the documentation provided by CBPDO to see where IBM.HINZ210.F2 is located on the CBPDO tape), **filevol** is the volume serial of the DASD device where the downloaded files reside, **jcl-library-name** is the name of the output data set where the sample jobs will be stored, **dasdvol** is the volume serial of the DASD device where the output data set will reside, and **indd** is either TAPEIN or FILEIN depending on your input DD statement.

Step 3: Modify sample JCL

These are the sample JCL that will be on your driving system. See Table 7-2 for the list. You can copy these JCL on you own library. Then modify them to fit your system settings.

Table 7-2 Sample JCL: Edit and submit

Job Name	Job Type	Description	RELFILE
INZALA	SMP/E	Sample job to allocate and initialize a new SMP/E CSI data set **(Optional)**	IBM.HINZ210.F2
INZALB	SMP/E	Sample job to allocate SMP/E data sets **(Optional)**	IBM.HINZ210.F2
INZRECEV	RECEIVE	Sample RECEIVE job	IBM.HINZ210.F2
INZALLOC	ALLOCATE	Sample job to allocate target and distribution libraries	IBM.HINZ210.F2
INZDDDEF	DDDEF	Sample job to define SMP/E DDEFs	IBM.HINZ210.F2
INZAPPLY	APPLY	Sample APPLY job	IBM.HINZ210.F2
INZACCEP	ACCEPT	Sample ACCEPT job	IBM.HINZ210.F2

Step 4: Allocate SMP/E CSI (optional)

If you are using an existing CSI, do not execute this job. If you are allocating a new SMP/E data set for this install, edit, and submit sample job INZALA to allocate the SMP/E data set for HPU.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Step 5: Initialize CSI zones (optional)

Edit and submit sample job INZALB to initialize SMP/E zones for HPU. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Step 6: Perform SMP/E RECEIVE

NOTE: If you obtained HPU as part of a CBPDO, use the RCVPDO job found in the CBPDO RIMLIB data set to RECEIVE the HPU FMIDs as well as any service, HOLDDATA, or preventive service planning (PSP) information included on the CBPDO tape. For more information, refer to the documentation included with the CBPDO.

Edit and submit sample job INZRECEV to perform the SMP/E RECEIVE for HPU. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code 0 if this job runs correctly.

Step 7: Allocate SMP/E target and distribution libraries and paths

Edit and submit sample job INZALLOC to allocate the SMP/E target and distribution libraries for HPU. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code 0 if this job runs correctly.

Step 8: Create DDDEF entries

Edit and submit sample job INZDDDEF to create DDDEF entries for the SMP/E target and distribution libraries for HPU. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code 0 if this job runs correctly.

Step 9: Perform SMP/E APPLY

When you perform an SMP/E apply, you are adding the source code of the program module in the target library. The distribution library will not be changed at this point. This would allow you to assess the impact done by the installation on your system without making permanent changes in our system.

But before you run the SMP/E accept job you must first do a requisite check-up on the system. So you should first do an APPLYCHECK before you do your SMP/E APPLY. The APPLY CHECK is done with the same job just that the CHECK parameter is present in the JCL. When this JCL is run with this parameter there will be no changes done but you will see the HOLDDATA or errors that prevented you from having a successful execution.

Edit and submit sample job INZAPPLY to perform an SMP/E APPLY CHECK for HPU. Consult the instructions in the sample job for more information. To receive the full benefit of the SMP/E Causer SYSMOD Summary Report, do *not* bypass the following on the APPLY CHECK: PRE, ID, REQ, and IFREQ. This is because the SMP/E root cause analysis identifies the cause only of **ERRORS** and not of **WARNINGS** (SYSMODs that are bypassed are treated as warnings, not errors, by SMP/E).

Once you have taken any actions indicated by the APPLY CHECK, remove the CHECK operand and run the job again to perform the APPLY, see Example 7-2.

Note: The GROUPEXTEND operand indicates that SMP/E apply all requisite SYSMODs. The requisite SYSMODS might be applicable to other functions.

Expected Return Codes and Messages from APPLY CHECK: You will receive a return code 0 if this job runs correctly.

Expected Return Codes and Messages from APPLY: You will receive a return code 0 if this job runs correctly.

Example 7-2 The SMP/E APPLY / APPLYCHECK JCL

```
//*************************************************************/
//*        INVOKE SMP/E                                       */
//*************************************************************/
//SMP     EXEC PGM=GIMSMP,PARM='DATE=U',REGION=4M
//SMPCSI   DD  DSN=#globalcsi,DISP=SHR
//SMPHOLD  DD  DUMMY
//SMPCNTL  DD  *
    SET BDY(targlib).               /* Set to TARGET zone      */
    APPLY  SELECT(                  /* APPLY for this FMID     */
                  HINZ210
                  )
    CHECK                           /* Do not update libraries */
    GROUPEXTEND                     /* APPLY all requis. PTFs  */
    FORFMID(HINZ210)
    BYPASS(HOLDSYS,HOLDUSER,        /* Bypass options          */
    HOLDCLASS(UCLREL,ERREL,HIPER)).
//*
/*
```

7.2.3 Customization procedures for HPU

After you finish installing the HPU, you need to customize some files to fit your system. The customization will allow you to recover values of variables from a previously installation. It also allows you to set the values of the variables of the JCL and data sets that will be used by the HPU for execution.

Step 1: Allocating the INZRSAVE library

Important: This step is required only for first time users of HPU.

Go to the ISPF panel P.3.2. Allocate a PDS for use by the INZT02 procedure, which stores a backup of the configuration data set (member INZTVAR).

This data set should have:

- RECFM=FB
- LRECL=80

Any value of BLKSIZE that is a multiple of the LRECL, SPACE=(TRK,(1,1,5)). The name of this library must be entered in member INZTDSN. See “Step 3: Editing the INZTDSN member” on page 128.

```
Data Set Information
Command ===>

Data Set Name  . . . : INZ.V2R1M0.SINZRSAV

General Data                          Current Allocation
 Volume serial . . . : SBOX38          Allocated tracks  . : 4
 Device type . . . . : 3390            Allocated extents . : 4
 Organization  . . . : PO              Maximum dir. blocks : 5
 Record format . . . : FB
 Record length . . . : 80
 Block size  . . . . : 27920          Current Utilization
 1st extent tracks . : 1               Used tracks . . . . : 4
 Secondary tracks  . : 1               Used extents  . . . : 4
                                       Used dir. blocks  . : 1
 Creation date . . . : 2002/10/08      Number of members . : 1
 Referenced date . . : 2002/10/09
 Expiration date . . : ***None***
```

Figure 7-1 Sample data set information for allocating the INZRSAVE library

Step 2: Executing the INZT01 procedure

The INZT01 is a REXX procedure that lets you retrieve customized values from a previous installation (from a previous INZRSAVE member) and then generate member INZTVAR in the SINZSAMP library.

You can execute this REXX procedure by going to the P.3.4 ISPF panel and typing ‘exec’ on the line beside it. See Figure 7-2

First time installation

For a first time installation, do the following:

1. Execute the INZT01 procedure located in the SINZSAMP library.
2. Do not enter anything in any fields on the displayed panel.
3. Put the cursor on the highlighted GENER field and press Enter, o type GENER

in the command field and press Enter.The generation process creates a member called INZTVAR in the SINZSAMP library.The response “Generation was OK” will be returned.

Reinstallation

If you have already customized a previous version of the product, do the following:

1. Execute the INZT01 procedure located in the SINZSAMP library.
2. Enter the name of the previous INZRSAVE data set in the field “Old file of variables to be retrieved” in the format *dsname(member)*.

3. Press Enter. Wait until “Select other files or start the generation” is displayed before doing the next step.
4. Put the cursor on the highlighted GENER field and press Enter, o type GENER in the command field and press Enter. The generation process creates a member called INZTVAR in the SINZSAMP library.The response “Generation was OK” will be returned.

Some of the variables in this member will already contain values retrieved from the INZRSAVE member of the previous installation.

```
 Menu  Functions  Confirm  Utilities  Help
_____________________________________________________________________________
EDIT              INZ.V2R1M0.SINZSAMP                      Row 00022 of 00033
Command ===> _____________________________________________ Scroll ===> CSR
            Name     Prompt       Size   Created          Changed          ID
_________ INZP01
_________ INZRECEV
_________ INZRSAVE
_________ INZRSKEL
_________ INZSWORK
_________ INZTDSN                  23  2002/10/08  2002/10/08 20:05:28  PAOLOR2
_________ INZTTBL
_________ INZTVAR                 333  2002/10/08  2002/10/09 12:24:24  PAOLOR2
__exec___ INZT01
_________ INZT02                  724  2002/10/08  2002/10/08 19:08:31  PAOLOR2
_________ INZUTIL                 169  2002/10/09  2002/10/09 12:24:46  PAOLOR2
_________ INZUTISK
          **End**
```

Figure 7-2 Executing the INZT01 REXX program

To verify if your job ran successfully go to the SINZAMP library and look for the INZTVAR. This member appears after the successful run of the INZT01.

Step 3: Editing the INZTDSN member

Edit member INZTDSN from the SINZSAMP library. Enter the names of the libraries that will contain the customized members generated by the INZT02 procedure. Replace `??HLQ??` in Example 7-3 with the prefix of the library where HPU has been installed. A description of the libraries is listed in Table 7-3.

Table 7-3 Description of libraries used by HPU

Variables in INZTDSN	Description
INZRSAVE	Specifies the dsname of the library that will contain the backup of the variables data set.This library was created in “Step 1.Allocating the INZRSAVE library
INZSAMP	Specifies the dsname of the library that will contain the customized installation JCL.Usually the SINZSAMP library is used.

Variables in INZTDSN	Description
INZPLIB	Specifies the dsname of the library that will contain the customized PARMLIB member(s).
INZCLST	Specifies the dsname of the library that will contain the customized CLIST procedures.
INZSLIB	Specifies the dsname of the library that will contain the customized ISPF skeletons.

Example 7-3 INZTDSN JCL — Replace the '??hlq??' with your high level qualifier

```
//*************************************************************************
//**
//*ENTER BELOW THE DSNAMES OF THE FILES THAT WILL CONTAIN THE *
//*CUSTOMIZEDMEMBERS *
//**
//*************************************************************************
//*
//*LIBRARY THAT CONTAINS THE BACKUP OF THE VARIABLES FILE
//INZRSAVE ??HLQ??.SINZRSAVE
//*
//*SAMPLE LIBRARY
//INZSAMP ??HLQ??.SINZSAMP
//*
//*LIBRARY FOR DB2 UNLOAD PARMLIB
//*NOTE:IT CAN BE THE SAME LIBRARY AS THE SAMPLE LIBRARY ABOVE
//INZPLIB ??HLQ??.SINZSAMP
//*
//*CLIST LIBRARY
//INZCLST ??HLQ??.SINZCLST
//*ISPF SKELETONS LIBRARY
//INZSLIB ??HLQ??.SINZSLIB
```

Note: Depending on your installation's standards, this library can be the SINZSAMP library or any library chosen to contain HPU's PARMLIB members. The same library should be used to set variable VIZ007 in the INZTVAR member.

Step 4: Editing the INZTVAR member

Member INZTVAR in the SINZSAMP library contains the list of variables that are to be customized. You may need to refresh the list of members in the SINZSAMP library to see the INZTVAR member. INZTVAR was created when the INZT01 procedure was executed.

This member contains the definition of the variables used to create the installation JCL and the INZUTIL member that contains the product parmlib.

This member is the input of the INZ02 job that has to be run after this member is edited.

If you encounter any error in the parameters of the installation JCL generated by these members, you should not change the variables in the JCL directly. You must go back to the INZTVAR or INZTDSN and change the values there then rerun the INZ02 or INZ01 job.

Any manual modification on the installation JCL generated by the INZ02 will be lost once the INZ02 is rerun.

In our example, 'INZ.V2R1M0' is the high level qualifier of the installation data sets. a sample of the JCL portion that needs to be modified is in Example 7-6 on page 132. The JCL

example below is the portion of the INZTVAR where the DB2 parameters are coded. The JCL that was generated has some default values in it. It is optional to change these values. However, the parameters that point to the DB2 libraries are required to be filled up.

Tip: If you ned to know the high level qualifiers of the DB2 libraries that you are using, you can see them in the job that started DB2 Master address space. You can view this in the SDSF panel Active Users (DA option). Look for the jobname with MSTR suffix. You can do this by typing ‘pre *MSTR’ inside the DA panel command line. Then open the job that has your subsystem name on as the prefix.

A description of the default JCL parameters

These JCL parameters are the ones that will be used as input by the INZ02 REXX procedure which you will execute in the next step. These parameters allow you to define your job card that will be used by all of the installation JCL, see Example 7-4.

Example 7-4 INZTVAR JCL — Parameter VIM101 to VIM105

```
*********************************************************************
*
****************** DEFAULT JCL PARAMETERS ***************************
*
* JOB CARDS FOR JCL
* (UP TO 5 LINES, BETWEEN THE COL=19 AND THE COL=72)
*                ***************************************************
  VIM101          //PAOLOHPU  JOB PAOLOR21,'DB2 UNLOAD',
  VIM102          //          MSGCLASS=A,CLASS=A,NOTIFY=&SYSUID,
  VIM103          //          REGION=0M
  VIM104          //*
  VIM105          //*
*                 ***************************************************
```

Table 7-4 describes the JCL parameters to be defined by the user.

Table 7-4 JCL parameters to be defined

Parameters	Description
VIM101 to VIM105	Enter your job card. Ensure that you include a region size on the job card; failure to do so can result in abend s106 when running HPU interactively.
VIM111	Specifies the name of the default unit to be used to allocate temporary data sets.
VIM112	Specifies the name of the default SYSOUT CLASS to be used in the customization JCL.

Dsnames of HPU libraries

Example 7-5 shows a listing of the libraries that HPU uses. These data sets names are specified in the INZTVAR module and it will be used as input by the INZ02 REXX procedure.

Example 7-5 INZTVAR JCL — HPU libraries

```
****************** DSNAMES OF THE PRODUCT'S LIBRARIES *****************
*
* LOAD MODULES LIBRARY (SINZLOAD)
  VIZ003        INZ.V2R1M0.SINZLOAD
*
```

```
* APF LOAD MODULES LIBRARY (SINZLINK)
  VIZ004 LIBUTIL   INZ.V2R1M0.SINZLINK
*
* PARMLIB LIBRARY (INZPLIB)
* BY DEFAULT USE THE SINZSAMP LIBRARY
* SHOULD BE THE SAME AS THE LIBRARY DESCRIBED IN THE INZTDSN MEMBER
  VIZ007           INZ.V2R1M0.SINZSAMP
*
* PRODUCT'S DBRM LIBRARY (SINZDBRM)
  VIZ012         INZ.V2R1M0.SINZDBRM
*
* ISPF LOAD MODULES LIBRARY (SINZLLIB)
  VIZ013       INZ.V2R1M0.SINZLLIB
*
* ISPF MESSAGES LIBRARY (SINZMLIB)
  VIZ015       INZ.V2R1M0.SINZMLIB
*
* ISPF PANELS LIBRARY (SINZPANL)
  VIZ016       INZ.V2R1M0.SINZPANL
*
* ISPF SKELETONS LIBRARY (SINZSLIB)
  VIZ017       INZ.V2R1M0.SINZSLIB
*
* ISPF TABLES LIBRARY (SINZTLIB)
  VIZ018       INZ.V2R1M0.SINZTLIB
```

Table 7-5 describes the parameters pointing to the HPU libraries.

Table 7-5 Data set parameters of the HPU library

Parameters	Description
VIZ003	Specifies the dsname of the LOAD MODULES LIBRARY (SINZLOAD).
VIZ004	Specifies the dsname of the APF MODULES LIBRARY (SINZLINK).
VIZ007	Specifies the dsname of the PARMLIB LIBRARY (INZPLIB). **Note:** This library should be the same as the one specified in the INZTDSN member (see “Step 3. Editing the INZTDSN member”).
VIZ012	Specifies the dsname of the DBRM LIBRARY (SINZDBRM).
VIZ013	Specifies the dsname of the ISPF load module library (SINZLLIB), which contains the load modules for the interactive application.
VIZ015	Specifies the dsname of the ISPF messages library (SINZMLIB), which contains the messages for the interactive application.
VIZ016	Specifies the dsname of the ISPF panels library (SINZPLIB), which contains the panels for the interactive application.
VIZ017	Specifies the dsname of the ISPF skeletons library (SINZSLIB), which contains the skeletons for the interactive application.

Parameters	Description
VIZ018	Specifies the dsname of the ISPF tables library (SINZTLIB), which contains the tables for the interactive application.

Example 7-6 shows how to define the DB2 libraries and subsystem name. These values are required to be edited. The values in italics are the ones that needs to be supplied by the user. These values depend on the DB2 set-up where HPU is being installed.

Example 7-6 INZTVAR JCL — Defining the DB2 libraries and subsystem name.

```
****************** COMMON DB2 PARAMETERS ******************************
*
* NOTE:   IF YOU WANT TO INSTALL THE PRODUCT ON SEVERAL DB2 SYBSYSTEMS
*         YOU SHOULD DUPLICATE THE DEFINITION OF SOME VARIABLES,
*         ONE FOR EACH SUBSYSTEM.
*         BE CAREFUL TO CODE THE VARIABLES THAT CORRESPOND TO THE
*         DIFFERENT SUBSYSTEMS IN THE SAME ORDER FOR EACH VARIABLE.
*
* DB2 SUBSYSTEMS
  VZD001          DB2G
* VZD001          LINE TO DUPLICATE TO SPECIFY OTHER DB2 SUBSYSTEM
*
* LIBRARY WHICH CONTAINS DB2 LOAD-MODULES (DSNLOAD)
  VZD003          DB2G7.SDSNLOAD
* VZD003          LINE TO DUPLICATE FOR DSNLOAD OF OTHER DB2 SUBSYSTEM
*
* LIBRARY WHICH CONTAINS DB2 RUNLIB LOAD-MODULES (DSNTIAD)
  VZD004          DB2V710G.RUNLIB.LOAD
* VZD004          LINE TO DUPLICATE FOR RUNLIB OF OTHER DB2 SUBSYSTEM
*
* PLANS CORRESPONDING TO DSNTIAD PROGRAM
  VZD005          DSNTIA71
* VZD005          LINE TO DUPLICATE FOR DSNTIAD PLAN OF OTHER DB2
*
*
* DB2 DSNEXIT LIBRARY
  VZD007          DB2V710G.SDSNEXIT
* VZD007          LINE TO DUPLICATE FOR DSNEXIT OF OTHER DB2 SUBSYSTEM
*
*
* IBM CONVERSION SERVICE LOAD LIBRARY
* LOAD LIB THAT CONTAINS CUNLCVN (IBM DOCNUM GI10-9760)
* (OPTIONAL)
  VZM006 SCUNMOD
*
```

Table 7-6 describes the most common DB2 parameters.

Table 7-6 Common DB2 parameters:

Parameters	Description
VZD001	Specifies the names of all DB2 subsystems on which the product is to be installed, or the DB2 group attachment name when processing inside a SYSPLEX data sharing environment.The name you specify should correspond to the value you specify for system/group in the PARM field of the EXEC statement. **Note**: You can specify several DB2 subsystems.To define more than one DB2 subsystem, blank out the asterisk (“*”)in column 1,and specify the DB2 name in column 19.
VZD003	Specifies the name of the library that contains the DB2 Load modules. Specify one value for each DB2 subsystem defined with variable VZD001.
VZD004	Specifies the name of the DB2 RUNLIB library that contains DSNTIAD. Specify one value for each DB2 subsystem defined with variable VZD001.
VZD005	Specifies the name of the plan that corresponds to the DSNTIAD program. Specify one value for each DB2 subsystem defined with variable VZD001.
VZD007	Specifies the name of the DB2 DSNEXIT LIBRARY.You should specify one value for each DB2 subsystem defined with variable(s)VZD001.
VZM006/SCUNMOD	Optional, it specifies the name of the IBM Conversion Service Load Library. If you want to perform conversions that imply CCSIDs that are non-SBCS or pairs of CCSID SBCS that are not supported by the SYSSTRINGS catalog table you must first install IBM OS/390 Support for Unicode. For more information on this program, see *OS/390 Support for Unicode Program Directory* and *OS/390 Support for Unicode: Using Conversion Service*s
VUX004/LOWMEM	Specifies the maximum number of active sorts running in the same step.The IBM Sort product supports a MAXSORT value of 1 through 9,as all of its modules are reusable. Most of the non-IBM sort products contain modules that require MAXSORT=1; this is because not all of these modules are reusable.
VUX005/MAXSORT	Specifies the memory size (below the 16 megabyte line) used by the sort program.

Parameters	Description
VUX006/WRKSPACE	Specifies an allocation percent for the sort input file, which can be one of the following: **PARTIAL** Corresponds to a primary allocation that is equal to 67 percent of the estimated size of the file to be sorted, and to a secondary allocation of 33 percent. **FULL** Corresponds to a primary allocation that is equal to 100 percent of the estimated size of the file to be sorted, and to a secondary allocation of 33 percent. HPU uses asynchronous DB2 STOP commands and checks whether they executed without error using DISPLAY commands. The following three variables are used to handle this process: WAITUNIT, WAITQTY, and WAITQTYM
VUX007/WAITUNIT	Specifies the wait time (1 unit = .01 second) between two unsuccessful tests of the STOP commands. The default value is 100 (1 second).
VUX008/WAITQTY	Specifies the number of times an unsuccessful STOP command will be tested before sending a WTOR to the console.If the operator answers CANCEL (“C”), HPU will stop with return code 8.If the operator answers WAIT (“W”), the wait process starts again. The default value is 20 times.
VUX009/WAITQTYM	Specifies the maximum wait time (in seconds) before sending an answer to the WTOR message. Utility execution will stop beyond this limit (return code 8).
VUX010/LIMUNIT	Specifies the maximum number of disk units to be used for allocation of a temporary work file. The default value is 9.
VUX019/WRKMXPR	Specifies the maximum size for the primary allocation of a work data set on DASD. When using very large work data sets, the primary allocation might be extended on several volumes according to the limit that was specified in the VUX010/LIMUNIT variable.In any case, the value that is provided for theVUX019/WRKMXPR variable must be lower than the capacity of the units that are used for these work data sets (VUM013),and the splitting up on these units must be taken into account.

HPU parameters

Table 7-7 describes the most common HPU parameters.

Table 7-7 Parameters for the HPU

Parameters	Description
VUM011/PLANOBJT	Specifies the plan name for the application.

Parameters	Description
VUM012	Specifies the name of the owner of the plan created for the product. Specify one value for each DB2 subsystem defined with variable VZD001.
VUX011	Specifies whether GRANT TO PUBLIC or TO USER (plan for the product) is to be done.

Specify one value for each DB2 subsystem defined with variable VZD001.

You can control the internal function parameters of the HPU by entering the values on the INZTVAR member. The JCL portion appears in Example 7-7.

Example 7-7 JCL portion where the HPU parameters are set

```
*
*APPLICATION PLAN FOR THE PRODUCT
VUM011 PLANOBJT ????????
*
*OWNER OF THE PLAN CREATEDFOR THE PRODUCT
VUM012 ????????
*VUM012 LINE TO DUPLICATE FOR PLAN OWNER IN OTHER DB2
*
*PUBLIC OR USER (GRANT ON THE PLAN CREATEDFOR THE PRODUCT)
VUX011 ??????
*VUX011 LINE TO DUPLICATE FOR OTHER DB2 SUBSYSTEM
*
*UNIT NAME FOR ALLOCATION OF TEMPORARY DATASETS
/VUM013 WRKUNIT &VIM111
*
*VOLUME(S)FOR ALLOCATION OF TEMPORARY DATA SETS
*(OPTIONAL)
VUM018 WRKVOL
*
*TAPE UNIT WHERE THE WORK DATASETS MUST BE ALLOCATED
*(OPTIONAL)
VUA007 WRKTUNIT
*
*MAXIMUM SIZE FOR WORK DATASET ON DASD (IN KILOBYTES)
VUX016 WRKUNTSW
*
*MAXIMUM NUMBER OF UNIT FOR TAPE TEMPORARY DATASET
VUX017 MAXTUNIT
*
*LIMIT NUMBER OF MESSAGES THAT ARE ISSUEDIF ANY ERROR
*CONCERNING THE STRUCTURE IS ENCOUNTEREDWHILE READING THE
*ROWS OF A TABLESPACE
*(OPTIONAL)
*VUX018 LDSERRLM
*
*QUIESCE OF SYSDBASE AND DBD01 FOR THE BATCH UTILITIES
*(YES/NO/OFF/FORCE)
VUM014 QUIESCAT YES
*
*USER USEDTO QUIESCE THE CATALOG TABLESPACES
*(INSTALL_SYSOPR/CURRENT_USER/USER)
VUM020 QUIESUSR INSTALL_SYSOPR
*VUM020 LINE TO DUPLICATE FOR OTHER DB2 SUBSYSTEMS
```

Table 7-8 shows the description of the most important parameter for HPU

Table 7-8 HPU general parameters (INZTVAR member)

Parameters	Description
VUM013/WRKUNIT	Specifies the name of the unit used to allocate temporary data sets.
VUM018/WRKVOL	Specifies the name of the volume where temporary data sets will reside.
VUA007/WRKTUNIT	Specifies the name of the tape unit that is used to allocate temporary files. When using temporary files on tape, specify a tape unit here.If no value is specified, the utility allocates its temporary files on the unit specified in the WRKUNIT parameter.
VUX016/WRKUNTSW	Specifies a threshold size (in kilobytes) for work data sets.All work data sets that exceed this threshold size will be allocated on the unit specified within the VUA007/WRKTUNIT parameter.
VUX017/MAXTUNIT	Specifies the maximum number of tape units that are provided for work data sets being used by a HPU job.
VUX018/LDSERRLM	Specifies the maximum number of messages that are issued if any error concerning the row structure is encountered while reading the rows of a table space thus allowing a limitation of the number of messages that are written into the spool.

Parameters	Description
VUM014/QUIESCAT	Specifies whether a quiesce point is to be taken on the following table spaces before running the utility: DSNDB06.SYSDBASE DSNDB01.DBD01 DSNDB06.SYSGROUP DSNDB06.SYSDBAUT DSNDB06.SYSUSER DSNDB06.SYSCOPY DSNDB06.SYSVIEWS DSNDB06.SYSSTATS YES A quiesce point is to be taken at execution time unless keyword QUIESCECAT NO was specified in the SYSIN of HPU. NO A quiesce point is NOT to be taken at execution time unless keyword QUIESCECAT YES was specified in the SYSIN of HPU. FORCE A quiesce point is ALWAYS to be taken at execution time, even if keyword QUIESCECAT NO was specified in the SYSIN of HPU. OFF A quiesce point is NEVER to be taken at execution time, even if keyword QUIESCECAT YES was specified in the SYSIN of HPU. *Default: NO* if this variable is blanked out
VUM020/QUIESUSR	Specifies the user who will run QUIESCE on the DB2 catalog table spaces.
INSTALL_SYSOPR	The user defined as SYSOPR when the product was installed will be used to run QUIESCE on the DB2 catalog table spaces.
CURRENT_USER	The user submitting the job will be used to run QUIESCE on the DB2 catalog table spaces.
USER name	The user name specified (up to 7 characters) will be used to run QUIESCE on the DB2 catalog table spaces. *Default: INSTALL_SYSOPR*
VUM022/QSBUFNO	Specifies the BUFNO for sequential QSAM (parameter BUFNO of the DCB for QSAM).
VUM023/VSBUFND	Specifies the BUFND for sequential VSAM (parameter BUFND of the ACB for VSAM).
VUM024/SRTVNBRE	Specifies the number of records in the sort work areas.
VUM025/SRTVSMIN	Specifies the minimum size (in bytes) of the sort work areas.

Parameters	Description
VUM026/SRTVSMAX	Specifies the maximum size (in bytes) of the sort work areas.

You can leave the HPU parameters as they are in Table 7-9, or you can customize them based on your installation's requirements. The JCL portion is reported in Example 7-8.

Example 7-8 JCL portion from the INZTVAR member.

```
*------------------PARAMETERS FOR DB2 UNLOAD -------------
*
*OPTION TO PERFORM "SELECT"STATEMENTS VIA DB2 WHEN THEY ARE
*NOT SUPPORTEDBY DB2 UNLOAD
VUU011 ULSEDB2 YES
*
*OPTION TO LOCK THE TABLESPACE
VUU012 ULLOCK NO
*
*OPTION TO QUIESCE THE TABLESPACE
VUU013 ULQSCE NO
*
*VALUE OF THE NULL INDICATOR
VUU014 ULNULL FF00
*
*CONVERSION OF DATE TYPE COLUMNS
VUU015 ULDATE DATE_C
*
*CONVERSION OF TIME TYPE COLUMNS
VUU016 ULTIME TIME_A
*
*CONVERSION OF TIMESTAMP TYPE COLUMNS
VUU017 ULTMSTP TIMESTAMP_B
*
*DISPLAY,POSITION OF THE SIGN AND DECIMAL SEPARATOR
VUU018 ULPIC
*
*OPTIONS OF THE GENERATEDLOADSTATEMENT AT THE TABLESPACE LEVEL
VUU019 ULOPTLDT (LOG(NO),NOTIFY(YES),ENFORCE(NO),SORTKEYS(&SORTKEYS))
*VUU119
*VUU219
*VUU319
*VUU419
*
*OPTIONS OF THE GENERATEDLOADSTATEMENT AT THE PARTITION LEVEL
VUU020 ULOPTLDP (RESUME(YES))
*VUU120
*VUU220
*VUU320
*VUU420
*
*DEGREE OF PARALLEL PROCESSING
VUU021 ULDEGREE ANY
*
*POSITION FOR NULL INDICATOR
VUU022 NULLPOS BEFORE
*
*DEFAULT SCHEME FOR UNLOAD TABLESPACE
*(EBCDIC/ASCII/UNICODE/ASIS)
VUU023 UNLSCHEM EBCDIC
```

```
*
*RETURN CODE IF ZERO LINE IS UNLOADED
VUU024 UNLZLRC 4
*HIGH LEVEL QUALIFIER OF DB2 Admin DATASETS
*EXAMPLE:DBTOOL
VUU025 ADB
*
*LIBRARY WHICH CONTAINS DB2 Admin COMMANDS TABLES
*EXAMPLE :DBTOOL.SADBTLIB
VUU026 ADB.V4R1M0.SADBTLIB
*
*LIBRARY WHICH CONTAINS THE ADBDMTI EXEC
*EXAMPLE :DBTOOL.SADBEXEC
VUU027 ADB.V4R1M0.SADBEXEC
```

Table 7-9 HPU parameters

Parameters	Description
VUU011/ULSEDB2	Specifies whether to process SELECT statements using DB2 when the statements are not supported by HPU. NO SELECT statements not supported by HPU will not be processed by DB2. YES SELECT statements not supported by HPU will be processed by DB2. *Default: YES*
VUU012/ULLOCK	Specifies whether to LOCK the tables of the table space. NO Tables of the table space are not to be locked. YES Tables of the table space are to be locked. *Default: NO*
VUU013/ULQSCE	Specifies whether to QUIESCE the table space. NO The table space is not to be quiesced. YES The table space is to be quiesced. *Default: NO*

Parameters	Description
VUU014/ULNULL	Lets you modify the null or not null indicator in USER FORMAT. OFF The null indicator is not present in the output data set.hhhh The first two digits (one hexadecimal character) represent the null indicator for a null column. The last two digits (one hexadecimal character) represent the null indicator for a not null column.
VUU015/ULDATE	Specifies the default conversion type for a date column when using FORMAT USER. For information about conversion types and FORMAT USER
VUU016/ULTIME	Specifies the DEFAULT conversion type for a time column when using FORMAT USER. For information about conversion types and FORMAT USER
VUU017/ULTMSTP	Specifies the default conversion type for a timestamp column when using FORMAT USER. For information about conversion types and FORMAT USER

Parameters	Description
VUU018/ULPIC	Defines the numeric data display format.The ULPIC parameter has three subparameters.All three must be specified. The first subparameter specifies the rules to print the sign. **'+'**Specifies that '+'is used for positive values and '-'is used for negative values. **'-'**Specifies that '-' is used for negative values and 'blank' is used for positive values. 'P' Specifies that the padding character is used for positive values and '-' is used for negative values. *Default: '-'* The second subparameter specifies the rules to position the sign. LEAD (default value) The sign will be placed in front of the numeric value.(LEAD is ignored for floating point numbers). TRAIL The sign will be placed after the numeric value.(TRAIL is ignored for floating point numbers.) The third subparameter specifies the decimal separator. **'.'**Default. Write a point as the decimal separator. **','**Write a comma as the decimal separator. *Default:* The default value is ('-',LEAD,'.'),meaning that the sign will be printed in front of the numeric value, the sign will only be shown for negative values, and the decimal separator will be a point.

Parameters	Description
VUU019/ULOPTLDT	Defines the parameters of the LOAD statement generated at the table space level. All subparameters must be enclosed in parenthesis as shown: SORTDEVT(SYSDA)SORTNUM(32) To code the Load options on several lines, use the following rules: 1.Remove the "*"(asterisk) in column 1 for the added lines that are used. 2.Enter the additional data starting in column 19. 3.The first string must begin with a left parenthesis, and the corresponding right parenthesis MUST be coded only on the last line used. For example: VUU019 ULOPTLDT (INDDN(SYSREC0),RESUME(NO), VUU119 REPLACE,KEEPDICTIONARY, VUU219 LOG(NO),ENFORCE(NO)) *VUU319 A list of parameters that are accepted follows: COPYDDN DISCARDDN DISCARDS ENFORCE(CONSTRAINTS/NO) INDDN KEEPDICTIONARY LOG(YES/NO) RECOVERYDDN REPLACE RESUME(YES/NO) SORTDEVT SORTKEYS (see note below) SORTNUM **Note:** If you specify the SORTKEYS keyword, the value you specify for the &SORTKEYS variable will be substituted with a value calculated according to the number of unloaded records.
VUU020/ULOPTLDP	Defines the parameters of the LOAD statement generated at the partition level. To code the Load options on several lines, use the following rules: 1.Remove the "*"(asterisk) in column 1 for the added lines that are used. 2.Enter the additional data starting in column 19. 3.The first string must begin with a left parenthesis, and the corresponding right parenthesis MUST be coded only on the last line used. A list of parameters that are accepted follows: RESUME(YES/NO) REPLACE KEEPDICTIONARY

Parameters	Description
VUU021/ULDEGREE	Defines the degree of parallelism, that is, the number of parallel tasks or I/O operations that DB2 can use to extract data from a partitioned table space. 1 Parallelism is not to be used. ANY The product is to decide whether parallelism will be used. *Default: ANY*
VUU022/NULLPOS	Specifies the position of the NULL indicator within the HPU output data sets: AFTER The NULL indicator will be set *after* the column data. BEFORE The NULL indicator will be set before the column data. *Default: BEFORE*
VUU023/UNLSCHEM	Defines the unload format for the data. **ASCII** Indicates that the unloaded data must be in ASCII format.HPU uses the subsystem's ASCII CCSID, unless overridden by specifying the CCSID option. **ASIS** Indicates that the data is unloaded in its original format.If the specification for the underlying table space cannot be determined (for example, if the data is processed by DB2), the CCSID returned by a standard prepare statement in SQLDA is used. You can also override ASIS by specifying the CCSID keyword. Specifying ASIS does not mean that no conversion is necessary. Conversion may still be required in some situations. **EBCDIC** Indicates that the data is unloaded in EBCDIC format.HPU uses the subsystem's EBCDIC CCSID, unless you override it by specifying the CCSID keyword. **UNICODE** Indicates that the data is unloaded in UNICODE format.HPU uses the subsystem's UNICODE CCSID, unless you override it by specifying the CCSID option. *Default: EBCDIC*

Parameters	Description
VUU024/UNLZLRC	Specifies the return code that is issued by HPU if zero rows are unloaded by at least one of the SELECT statements of the UNLOAD command. *Default:4*
VUU025	Specifies the high-level qualifier of the DB2 Administration tool libraries. This information is used by the sample programs INZADBI and INZDB2IX (in the SINZSAMP library) to update the DB2 Admin tool and the Data Management tools Launchpad tables.
VUU026	Specifies the name of the library that contains the DB2 Administration tool command tables. This information is used by the sample programs INZADBI and INZDB2IX (in the SINZSAMP library) to update the DB2 Administration tool and the DB2 tools Launchpad tables.
VUU027	Specifies the name of the library that contains the ADBMTI EXEC. This information is used by the sample programs INZADBI and INZDB2IX (in the SINZSAMP library) to update the DB2 Administration tool and the DB2 tools Launchpad tables.***JCL parameters:*** You can leave all variables as they are if you want to use the same job card and SYSOUT class for all generated JCL. Or you can change specifications on each job card.
VUX101 to VUX105,VUX111,VUX112	Specify the job parameters for the JCL that creates the plan for the utilities (INZBIND JCL) as well as the SYSOUT CLASSes of SYSTSPRT and SYSPRINT.
VUX141 to VUX145,VUX151	Specify the job parameters for the JCL that customizes the Load modules with the name of the HPU PARMLIB as well as the SYSOUT CLASS of SYSPRINT.

See Example 7-9 for more JCL parameters.

Example 7-9 JCL parameters in INZTVAR

```
*********************JCL PARAMETERS **********************************
*
*THE CUSTOMIZATION OF THE FOLLOWING VARIABLES CAN BE SKIPPEDIF
*YOU WANT TO USE THE SAME JOB CARDANDSYSOUT CLASS FOR ALL THE
*GENERATEDJCL.IN THIS CASE THE VALUES USEDWILL DEFAULT TO THE
*CONTENT OF THE VARIABLES VIM101 TO VIM112 DEFINED AT THE BEGINNING
*OF THIS MEMBER
*
*
*JOB PARAMETERS FOR THE JCL WHICH CREATE THE PLAN FOR UTILITIES
*(INZBINDJCL)
*JCL CARD
*****************************************************
/VUX101 &VIM101
```

```
/VUX102 &VIM102
/VUX103 &VIM103
/VUX104 &VIM104
/VUX105 &VIM105
*******************************************************
*SYSOUT CLASS OF SYSTSPRT
/VUX111 &VIM112
*SYSOUT CLASS OF SYSPRINT
/VUX112 &VIM112
*
*
*JOB PARAMETERS FOR THE JCL WHICH CUSTOMIZE THE LOADMODULES
*WITH THE NAME OF THE PARMLIB
*(INZPARM JCL)
*JCL CARD
*******************************************************
/VUX141 &VIM101
/VUX142 &VIM102
/VUX143 &VIM103
/VUX144 &VIM104
/VUX145 &VIM105
*******************************************************
*SYSOUT CLASS OF SYSPRINT
/VUX151 &VIM112
```

Note that you do not have to input all the parameters that appear in the job, you only need to change those values which the default is different from the value that you want. After you finish inputting the parameters, record the changes you made and then save the data.

Step 5: Executing the INZT02 procedure

After filling up the parameters in the INZTVAR member, you proceed in the customization by running the INZT02 REXX procedure. The INZT02 procedure lets you generate all JCL and data sets for HPU. This procedure creates or replaces members in the libraries that are defined in member INZTDSN. You execute this procedure by typing exec on the line that correspond to the INZT02 member inside the SINZAMP library. See Figure 7-3. This is inside the P.3.4 panel.

```
 Menu  Functions  Confirm  Utilities  Help
_____________________________________________________________________________
EDIT              INZ.V2R1M0.SINZSAMP                      Row 00022 of 00033
Command ===> _____________________________________________ Scroll ===> CSR
           Name     Prompt       Size   Created          Changed          ID
_________ INZP01
_________ INZRECEV
_________ INZRSAVE
_________ INZRSKEL
_________ INZSWORK
_________ INZTDSN                 23  2002/10/08  2002/10/08 20:05:28  PAOLOR2
_________ INZTTBL
_________ INZTVAR                333  2002/10/08  2002/10/09 12:24:24  PAOLOR2
_________ INZT01
__exec___ INZT02                 724  2002/10/08  2002/10/08 19:08:31  PAOLOR2
_________ INZUTIL                169  2002/10/09  2002/10/09 12:24:46  PAOLOR2
_________ INZUTISK
          **End**
```

Figure 7-3 Executing the INZT02 REXX program

The INZT02 procedure also creates a backup of member INZTVAR in member.

INZRSAVE in the library created in “Step 1: Allocating the INZRSAVE library” on page 126. This member is used by the installation process when installing a new version of the product.

Note: Be aware that execution of the INZT02 procedure will override any manually customized member that is supposed to be created by this procedure.

The list of members created by the INZT02 procedure is as follows:

In the SINZSAMP library:

- INZBIND
- INZPARM
- INZADBI
- INZDB21X

In the INZPLIB library or other library name assigned during customization.

- INZUTIL

For more information, see “Step 3: Editing the INZTDSN member” on page 128.

Step 6: Integrating HPU into DB2 tools Launchpad

You have the option to use the DB2 tools Launchpad to start the HPU. To do this you have to run the CLIST INZADBI. The parameters in the INZTVAR member (VUU025 and VUU027) will be used by the CLIST for this.

The display panel in Figure 7-4 will appear when you run the INZADZBI. Press Enter to confirm the new command HPU.

```
------------------DB2 Tools Table -ADD An Entry ---------------------14:30 ~
Command ===>









Prog No.:5655-I19 Release :210

Tool Name:HPU High Performance Unload

Code :HPU Group :200 Installed:Y

Command :SELECT CMD(INZHPU D LP )
```

Figure 7-4 Add an entry panel

When INZADBI completes successfully, a new line - HPU, will be added to the DB2 tools Launchpad.

Step 7: Integrating HPU into DB2 Admin tool

You can customize your system to run HPU interactively using the DB2 Administration tool. To do this, run the CLIST INZDB21X. The CLIST uses the value for VUU026 that you set previously in the INTZVAR member.

Messages will be displayed to inform you about the progress of the CLIST run. There will be requested actions along the duration of the run. These actions needs to be performed to continue the installation. After the successful completion of the CLIST run you start HPU interactively using the DB2 Admin tool. An example to run a CLIST program is in Figure 7-5. This is inside the P.3.4 panel

```
Menu  Functions  Confirm  Utilities  Help
_____________________________________________________________________________
EDIT              INZ.V2R1M0.SINZSAMP                      Row 00001 of 00033
Command ===> _____________________________________________ Scroll ===> CSR
           Name     Prompt       Size   Created          Changed          ID
_________ INZACCEP
_________ INZADBI                133  2002/10/09  2002/10/09 12:24:46  PAOLOR2
_________ INZADBSK
_________ INZALA
_________ INZALB
_________ INZALLOC
_________ INZAPPLY
_________ INZBIND                 31  2002/10/09  2002/10/09 13:00:03  PAOLOR2
_________ INZBNDSK
_________ INZDB2SK
__exec___ INZDB21X               387  2002/10/09  2002/10/09 12:24:46  PAOLOR2
_________ INZDDDEF
_________ INZEXECU                56  2002/10/09  2002/10/09 13:57:28  PAOLOR2
_________ INZEXIT
_________ INZHPJSK
_________ INZHPUSK
_________ INZIO0
_________ INZIO1
```

Figure 7-5 Executing the INZDB21X CLIST program

Step 8: Binding and granting the plan

Go to the SINZASMP library and look for the INZBIND member. This member was generated during the installation process. It contains JCL that should be used to bind and grant the plan for HPU. The generated JCL is customized to bind and grant the HPU plan on each DB2 subsystem defined by variable VZD001 in the INZTVAR member. The INZBIND JCL is shown in Example 7-10.

Proceed as follows:

1. Review the contents of member INZBIND in the SINZSAMP library to verify that the JCL reflects your environment.If you need to make changes, proceed to step 4.
2. Submit member INZBIND.
3. Check for correct execution.
4. If you have a problem with the BIND command or the GRANT statement, do not directly modify this member; instead, correct the invalid values in the INZTVAR member and re-execute the INZT02 procedure (see “Step 4.Editing the INZTVAR member” and “Step 5. Executing the INZT02 procedure’.)

Example 7-10 JCL to bind the internal plan of HPU

```
//PAOLOHPU JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=X,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*
//*******************************************************************
//* BIND THE INTERNAL PLAN OF THE PRODUCT                           *
//*******************************************************************
//EXECUTE  EXEC PGM=IKJEFT01
//STEPLIB  DD  DSN=DB2G7.SDSNLOAD,
//             DISP=SHR
```

```
//DBRMLIB  DD  DSN=INZ.V2R1M0.SINZDBRM,
//             DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTSIN  DD  *
    DSN SYSTEM(DB2G)
    BIND PLAN(DSNHPUL) DYNAMICRULES(RUN) -
    OWNER(MICHAEL) -
    MEMBER( -
      INZQUERY -
      INZXPR2 -
      INZACDB2 -
      )       -
    ACQUIRE(USE) RELEASE(COMMIT) ISOLATION(CS) VALIDATE(RUN)
    RUN PROGRAM(DSNTIAD)  PLAN(DSNTIA71) -
    LIB('DB2V710G.RUNLIB.LOAD')
    END
//SYSIN   DD *
SET CURRENT SQLID = 'MICHAEL';
GRANT EXECUTE ON PLAN DSNHPUL TO PUBLIC;
/*
```

Important: If you encounter an error in the INZBIND jcl (RC > 0). Do not edit this member directly. Instead change the INZTVAR member then rerun the INZT02.

Step 9: Submitting the INZPARM member

The INZPARM member was generated in the SINZSAMP library by the installation process. It contains JCL that should be submitted to insert the dsname of the HPU PARMLIB in the product's load modules.

Proceed as follows:

1. Review the contents of member INZPARM in the SINZSAMP library to verify that the JCL reflects your environment. If you need to make changes, proceed to step 4.
2. Submit member INZPARM.
3. Check for correct execution.
4. If you have a problem with execution of this job, do not directly modify this member; instead, correct the invalid values in the INZTVAR member and re-execute the INZT02 procedure (see "Step 4: Editing the INZTVAR member" on page 129 and "Step 5: Executing the INZT02 procedure" on page 145.)

Important: The HPU PARMLIB consists of member INZUTIL. INZUTIL is located in the library whose dsname is specified in variable INZPLIB in the INZTDSN member and in variable VIZ007 in the INZTVAR member.

Member INZUTIL, which was generated by the INZT02 procedure, should not be modified directly; instead, you should modify the INZTVAR member and re-execute the INZT02 procedure (see "Step 4.Editing the INZTVAR member" and "Step 5.Executing the INZT02 procedure".)

To see which variable of member INZTVAR corresponds to a specific PARMLIB variable, enter a FIND command on the PARMLIB variable's name in the INZTVAR member.

7.3 Data formats used by the HPU

In this section we describe the data formats in input and output supported by HPU.

7.3.1 Sources of input data that can be used by HPU

HPU can use the following sources for input data:

- Non partitioned table spaces
- Partitioned table spaces
- Full image copies
- Last full image copies
- Incremental image copies

Different input data will require different syntax to process.

Non-partitioned table space

HPU can use a non-partitioned table space as source of data. You can unload several tables at the same time as long as they belong to one non-partitioned table space. The unload requests are processed in parallel with each other.

Partitioned table space

A partitioned table space can contain only one table. It usually stores portions of a table that has more than 1 GB of data. HPU can work on all partitions or on a subset of partitions. The partitions can be processed by HPU concurrently.

Full image copy (FIC)

A full image copy is an exact reproduction of all or part of a table space. HPU can get its input data from a full image copy. In this case, the rules for processing are the same as those that apply to a table space:

- If the FIC is of the whole table space, HPU will process every partition
- If the FIC is of only one partition of the table space, then HPU will process the specific partitions contained in the FIC.

Last full image copy

You can also ask HPU to do the unload against the last FIC data set taken for a tablespace. DB2 keeps a record of the FIC data sets created in the SYSIBM.SYSCOPY system table. It automatically refers to this table to determine the last full image copy taken from the table.

7.3.2 Output data formats

There are two kinds of data unload: the physical unload and the logical unload. The physical unload involves unloading the entire table space. A table space is mapped to a physical entity such as a device or a file. Hence, the name ‘physical’ unload. The logical unload involves unloading data base objects like tables and views. Tables and views are logical entities in a data base. HPU has five types of data formats. The physical unload has one kind of output format white the logical unload has four kinds of output formats.

Physical unload

In doing a physical unload, you just have to specify the table space name. You do not have to code any SQL in the unload statement:

- Reorg unload-only

HPU lets you unload a table space in the same format as the IBM DB2 Reorg utility. In this case, the table space is unloaded regardless of the tables it contains.To do this, specify the UNLDDN parameter in the UNLOAD command. You cannot select the rows to unload or change the format of these rows except by using an exit. You can limit the number of rows unloaded and do some sampling on the unloaded rows. Be aware that the sampling is done before the call to the exit. Even if you use this output format alone, you can still code the SELECT statements.These SELECT statements will be processed in parallel, but you can have only one UNLDDN statement per execution.

Logical unload

A logical unload is done by coding one or more SELECT statements. Each SELECT statement needs to have its own OUTDDN data set to handle the result set. Logical unload enables you to filter the rows and columns to be unloaded, and specify the output format. These are the file formats that can be created in a logical unload:

DSNTIAUL

HPU provides a simplified way to produce output identical to that produced by the DSNTIAUL program. In this case, the SELECT statement can contain only a column name, and no format conversion is available. As an option, you can generate output based on a table other than the one you are unloading. This is done using the keyword LIKE. When using LIKE, the selected column must be compatible with the *like* table.If any conversion is required, it follows HPU format rules.

DELIMITED

A delimited file is characterized by delimited rows and character strings. To create this type of output just specify what kind of character you want to be used as a delimiter. The output can be ASCII or EBCDIC file.

VARIABLE

Specify FORMAT VARIABLE to indicate that the output data set must be compatible with the DB2 LOAD data set. The default output data set is variable blocked (VB), but you can specify another format (F, FB, or V) in the JCL. HPU determines the LRECL at run time using the following rules:

a. Fixed and fixed blocked: the LRECL must be larger than or equal to the sum of the lengths of the fields.

b. Variable: the LRECL must be larger than or equal to the sum of the lengths of the fields plus 4.

Variable-length fields are counted using their maximum length, plus 2 bytes for the length of the field.

You can specify the following options for formatting the variable columns:

– END

The characteristics and the sequence of fields in the generated data set correspond to those in the SELECT statement. The fields generated in the data set are also typical of those in DSNTIAUL format; the only differences are:

Columns in DATE, TIME, or TIMESTAMP have the format ISO, which means:

– DATE corresponds to format YYYY-MM-DD
– TIME corresponds to HH.MM.SS
– TIMESTAMP corresponds to YYYY-MM-DD-HH.MM.SS.NNNNNN

If a column accepts the nulls, the null indicator is generated in front of the field. This indicator contains the value X ’FF’ if the field is null and X ’00’ if the value is usable. If the last selected column is variable, the type of the default generated data set will be VB, and

this last column will be written only on its effective length; the two length bytes are placed before the column.

You can override the default DATE/TIME/TIMESTAMP format by specifying an options_block at the SELECT level.Only a SELECT level options_block will be taken into consideration for this format:

– ALL

The only difference between this format and the END format is that all the variable columns are written using their actual length.

– LIKE table-name

The use of FORMAT VARIABLE with a table name is the same as for FORMAT DSNTIAUL and the differences described above are still valid.

USER

Specify FORMAT USER to indicate that you want the unloaded data to be formatted according to the keywords specified in the user_block. You can change field attributes for all selected columns, which means you can specify several keywords for each column, according to the type of data the column contains.

The default values are determined by the values set in the options_block.

If all the fields unloaded are fixed, then the default RECFM is FB.If at least one output field is variable, then the default RECFM is VB.

If the LRECL is not specified, HPU determines it at run time using the following rules:

– Fixed

  The LRECL of the data set is equal to the sum of the maximum length of fields, regardless of what was specified as the LRECL in the JCL.The output data set will be in FB format.

– Variable and variable blocked

  The LRECL of the data set is equal to the sum of the maximum length of fields plus 4,regardless of what was specified as the LRECL in the JCL.The output data set will be in VB format.

You can customize every output column however you want.You can force the conversion between type, change the date or time format, add or remove a length field, add or remove a null indicator, justify the content left or right, select a padding character, select a delimiter character for date and/or time, and so on.

Data type conversion in FORMAT USER

HPU allows you to get unload data into several data types. You can change them or retain their original data type. The TYPE keyword of the SELECT statement (option block for FORMAT USER) lets you create several types of data in the output.These types are declared in the keyword TYPE.The use of this keyword implies that data is to be converted from the original column type to the type declared in the TYPE keyword.

The following subsections describe the output data types that are allowed.

– Numeric data

  • INTEGER or INT

  Whole numbers in a binary word of 31 bits plus the sign.

  • SMALLINT

  Whole numbers in a binary halfword of 15 bits plus the sign.

  • DECIMAL(n,m) or DEC(n,m)

Standard decimal value contained in (n/2+1)bytes.The default value is DECIMAL or DEC and is equivalent to DECIMAL (5,0).

- FLOAT(n)

Number is simple floating point precision if (0<n<22)in a fullword.Number is double floating point precision if (21<n<54).

Default: double precision

– Nonnumeric data

- CHARACTER(n)or CHAR(n)

Character string of length n (0<n<255)bytes

- VARCHAR(n)

A two-byte length field followed by n characters.The size equals n+2 bytes.The DB2 type LONG VARCHAR is not used in a sequential data set.

- GRAPHIC(n)

Graphic character string coded on 2n bytes.One character equals two bytes.

- VARGRAPHIC(n)

Variable-length graphic character string coded on 2n+2 bytes.The DB2 type

- LONG VARGRAPHIC is not used in a sequential data set.

Authorized conversions in FORMAT USER output

Table 7-10 shows all conversions allowed between DB2 data types and HPU data types.

Table 7-10 Data type conversions allowed in DB2

DB2 data type	Output data types
INTEGER	SMALLINT, DECIMAL, FLOAT, CHAR
SMALLINT	SMALLINT, INTEGER, CHAR, DECIMAL(p,q), FLOAT
DECIMAL(m,n)	SMALLINT, INTEGER, CHAR, DECIMAL(p,q), FLOAT
FLOAT	SMALLINT, INTEGER, CHAR, DECIMAL, FLOAT
CHAR(n)	CHAR(m), VARCHAR(m)
VARCHAR(n)	CHAR(m), VARCHAR(m)
LONG VARCHAR(n)	CHAR(m), VARCHAR(m)
GRAPHIC(n)	GRAPHIC(m), VARGRAPHIC(m), CHAR(m)
VARGRAPHIC(n)	GRAPHIC(m), VARGRAPHIC(m)
LONG VARGRAPHIC(n)	GRAPHIC(m), VARGRAPHIC(m)

Output encoding scheme

For the DSNTIAUL, DELIMITED, VARIABLE, and USER output formats, the translations from EBCDIC to ASCII and from ASCII to EBCDIC are supported only for single byte character set (SBCS) character strings. Translations are done using the translation tables in the SYSIBM.SYSSTRINGS table. Other types of translation can be done using Unicode Conversion Services.

7.4 Using HPU

The HPU is a powerful tool which is relatively easy to use. It is versatile in the sense that it can be used interactively and in batch mode. THe HPU for z/OS can unload data from a mainframe environment and make the output file in ASCII which makes it readable by distributed platforms (Windows, UNIX, and Linux.)

You can use the HPU either by running a TSO batch job or by running it interactively using the DB2 Administration tool or DB2 tools Launchpad. The data input can come from table spaces or image copies. The output can be in different formats such as delimited, variable or DSNTIAUL compatible. The HPU can also do the EBCDIC to ASCII conversion and vice-versa.

The authority and privilege needed for HPU are:

- If the unload is performed from an image copy, RACF READ authority is required on the image copy data set.
- DB2 SELECT privilege on the unloaded table or view, and DISPLAY privilege on the database.
- If QUIESCE=YES is specified:
  - IMAGECOPY privilege is required on the database.
  - If the table space is in copy pending, the user must be able to issue START and STOP on the database.
- If LOCK=YES is specified, the user must have SELECT privilege on all tables of the unloaded table space.

7.4.1 Using the HPU in batch mode

The syntax in using the HPU is similar to the syntax of the DB2 Unload utility. HPU can selectively unload rows by coding the SQL select statement in the UNLOAD statement. The full syntax of the SQL SELECT can be specified in the UNLOAD statement, however, for the statements that require more than one scan of the table space, HPU cannot directly process the SQL, and gives the user the option of invoking DB2 to process it; HPU will then take over to provide the specified output formatting. For statements that cannot be executed by HPU, standard DB2 SQL performance is expected.

Any SQL statement that requires more than one pass on the data to be processed will be passed to the DB2 engine. One example is table joins. HPU opens the data set and reads from the first page to the last page. If it needs to loop back in order to satisfy a query it will pass the control to the DB2 engine for processing.

A sample job to run the HPU

Example 7-11 is a HPU batch job in the INZEXECU member of the SINZSAMP library. This JCL can also be used as a template for other HPU jobs.

Example 7-11 The INZEXECU JCL sample for using the HPU

```
//PAOLOHPU JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//*                                                           *
//*        DB2 UNLOAD JCL                                     *
//*                                                           *
//*        IN THIS SAMPLE :                                   *
```

```
//*                                                              *
//*      - THE DB2 SUBSYSTEM IS                 DB2G             *
//*      - THE DB2 UNLOAD LOAD MODULES ARE                       *
//*        IN THE LOADLIB                       INZ.V2R1M0       *
//*      - THE EXECUTION REPORT WILL BE                          *
//*        WRITTEN ON THE DDNAME                SYSPRINT         *
//*                                                              *
//****************************************************************
//STEP1    EXEC PGM=INZUTILB, PARM='DB2G,DB2UNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//*
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT
      DB2 YES
      QUIESCE YES QUIESCECAT YES
      OPTIONS DATE DATE_A
       SELECT 1, 2, 3 FROM PAOLOR7.EMP
        ORDER BY 1, 2 DESC
        OUTDDN (UNLDDN1)
        FORMAT VARIABLE ALL

        SELECT 1, 4, 5 FROM PAOLOR7.DEPT
        OUTDDN (UNLDDN2)
        FORMAT DSNTIAUL
        LOADDDN LOADDDN1 LOADOPT (RESUME NO REPLACE)

/*
//SYSPRINT DD  SYSOUT=*
//*
//********* DDNAMES USED BY THE SELECT STATEMENTS **********
//*
//UNLDDN1  DD  DSN=PAOLOR2.MD.UNLD1,
//             DISP=(NEW,DELETE,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(10,20),RLSE)
//UNLDDN2  DD  DSN=PAOLOR2.MS.UNLD2,
//             DISP=(NEW,DELETE,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(10,20),RLSE)
//LOADDDN1 DD  DSN=PAOLOR2.MD.LOADA,
//             DISP=(NEW,DELETE,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(10,20),RLSE)
```

In the batch job shown in Example 7-11, HPU executes the unload of two different tables inside one tables space. There are two select statements here, one for each table, while the unload is issued at table space level. In this example, the data is unloaded from the DSNDB04.DEPT table space. This table space contains the EMP and DEPT tables. Two different tables can be unloaded at the same time if they belong to the same tablespace. The unloaded data is located in the specified OUTDDN. The OUTDDN points to another data set specified by the UNLDDN1 and UNLDDN2. The UNLDDN1 and UNLDDN2 specify the data set names where the table space is unloaded. They will contain the result set of the select statement that corresponds to them. As shown, each select statement needs its own OUTDDN. In this case, PAOLOR2.MD.UNLD1 and PAOLOR2.MS.UNLD2 are the data sets that will receive the unloaded data from the EMP and DEPT table correspondingly. We have chosen to have PAOLOR2.UNLD1 in variable format, and PAOLOR2.UNLD2 in DSNTIAUL format.

Note: You can unload tables from different table spaces in parallel. Just specify one select statement per table.

The EXEC statement

In the EXEC statement in our example HPU job example:

```
//STEP1    EXEC PGM=INZUTILB, PARM='DB2G,DB2UNLOAD'
```

The control program being executed by the job is the INZUTILB. The library where the INZUTILB is in should be APF authorized. In our case, it is in INZ.V2R1M0.SINZLINK library which we specified in the STEPLIB.

The PARM statement contains the DB2 subsystem name. The name can be either the name of a DB2 subsystem inside a non data sharing environment, or the DB2 group attachment name when processing inside a SYSPLEX data sharing environment. The name should correspond to the value in VZD001of INZTVAR. In our example, we use the DB2 subsystem name which is DB2G.

The unique identifier for your HPU job is specified in the second parameter of the PARM equation. This is UID. You cannot use special characters in the UID. In our example, the UID is DB2UNLOAD.

DDNAMES used in the HPU job

The DDNAMES used in the HPU are reserved and user specified.

The reserved DD names allocated dynamically by HPU are shown in Table 7-11. You cannot use these DDNAMES for other purpose.

Table 7-11 Reserved DDNAMES allocated dynamically by HPU

DDNAMES	Description
INFOL001	DD name used to save the SYSIN data set.
UTPRINT	Output data set for SORT utility messages.
SYSIN	Data set used for the call to the DSNUTILB program when HPU QUIESCEs the catalog.
SYSPRINT	Data set used for the call to the DSNUTILB program when a QUIESCE is needed.
SORT DDNAMEs	HPU calls the SORT utility when you specify an ORDER BY or an ORDER CLUSTER.Thus,SORT DDNAMEs are allocated dynamically.
INFOIC *nn*	DD name used to identify the last image copy, where *nn* is a sequence number.

There are also DDNAME in the HPU JCL the user needs to provide. These DDNAMES are shown in the Table 7-12.

Table 7-12 DDNAMES that the user provides

DDNAMES	Description
STEPLIB	HPU must be able to access the exits used by DB2 to ensure compatibility with products using these standard exits.Therefore,you must specify the location of libraries containing exits for DB2 and DSNHDECP for this DB2 subsystem in STEPLIB, if the libraries are not in the LINKLIST.
SYSIN	Data set containing commands for HPU.
SYSPRINT	DD name for the data set that receives the report from execution of HPU.
SYSTERM	DD name for the data set that receives the additional diagnostic information from execution of HPU.
INFPLIB	This is an optional DDNAME. It connects HPU to the PARMLIB that contains member INZUTIL. This ddname should be omitted if, during installation, member INZPARM was customized and submitted. You will encounter an error in your batch job if you have customized INZPARM and still coded this option in your JCL.
COPYDD	This is an optional DDNAME. Value of the COPYDDN parameter specified in the SYSIN. Specify a DD statement naming the image copy data set from which the unload is to be performed. To allow HPU to do processing in parallel for partitioned table spaces using image copies as input, code in your JCL one copydd *nnn* statement for each partition you want to unload (copydd *0*1 copydd *02* ...,copydd *nn*n ; *nnn* is a 2-o 3-digit sequential number.
OUTDD	This DDNAME is optional. Value of the OUTDDN parameter specified in the SYSIN. Specify a DD statement naming the data set that will contain the result of a SELECT (logical unload). For information on processing in parallel for partitioned table spaces, see OUTDDN description
UNLDD	This is an optional DDNAME. **unldd** is the value of the UNLDDN parameter specified in the SYSIN. Specify a DD statement naming the data set that will contain the physical unload of your table space.

DDNAMES	Description
LOADDD	This is an optional DDNAME. **loaddd** is the value of the LOADDDN parameter specified in the SYSIN. Specify a DD statement naming the data set that will contain the SYSIN for a LOAD allowing the RELOAD of a SELECT (in DSNTIAUL or VARIABLE format) into the same or another table. **Note:** If you are requesting that HPU do processing in parallel for partitioned table spaces, you MUST specify this processing for all ddnames (copydd, unldd, and outdd) coded in the JCL.

7.5 Components of the HPU statement

The HPU statement is composed of statement blocks. These blocks are joined together to form the HPU statement. Not all blocks are required. The blocks that you include in your statement depends on the requirement you have for the output file.

7.5.1 HPU blocks

The HPU statement is composed of a main block and four optional blocks, see Figure 7-6. Only the main block is required for the HPU statement. The four optional blocks could be incorporated in the statement depending on the intended output file.

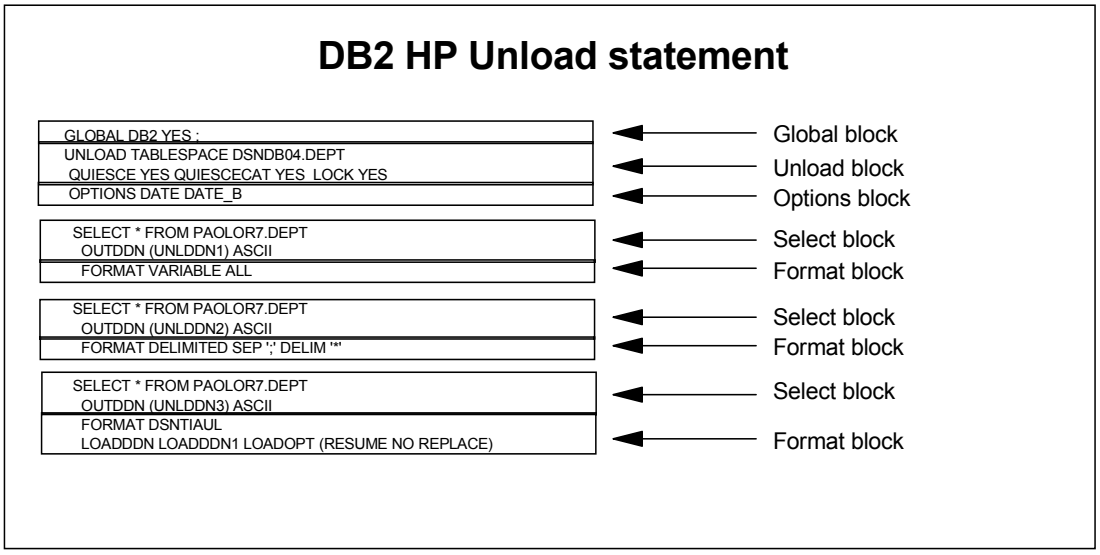

Figure 7-6 The HPU blocks

The Unload block is the main block. It is where all other blocks are connected. A Select block, a Global block, and/or an Options block can be connected to an Unload block. The Format block can only be connected to a Select block. If your HPU statement does not have a Select block then the Format block is not needed. The Options block can be connected to a Select block, Global block, and the Unload block, but it cannot be connected to a Format block. The Global block can contain an Options block. See Figure 7-7.

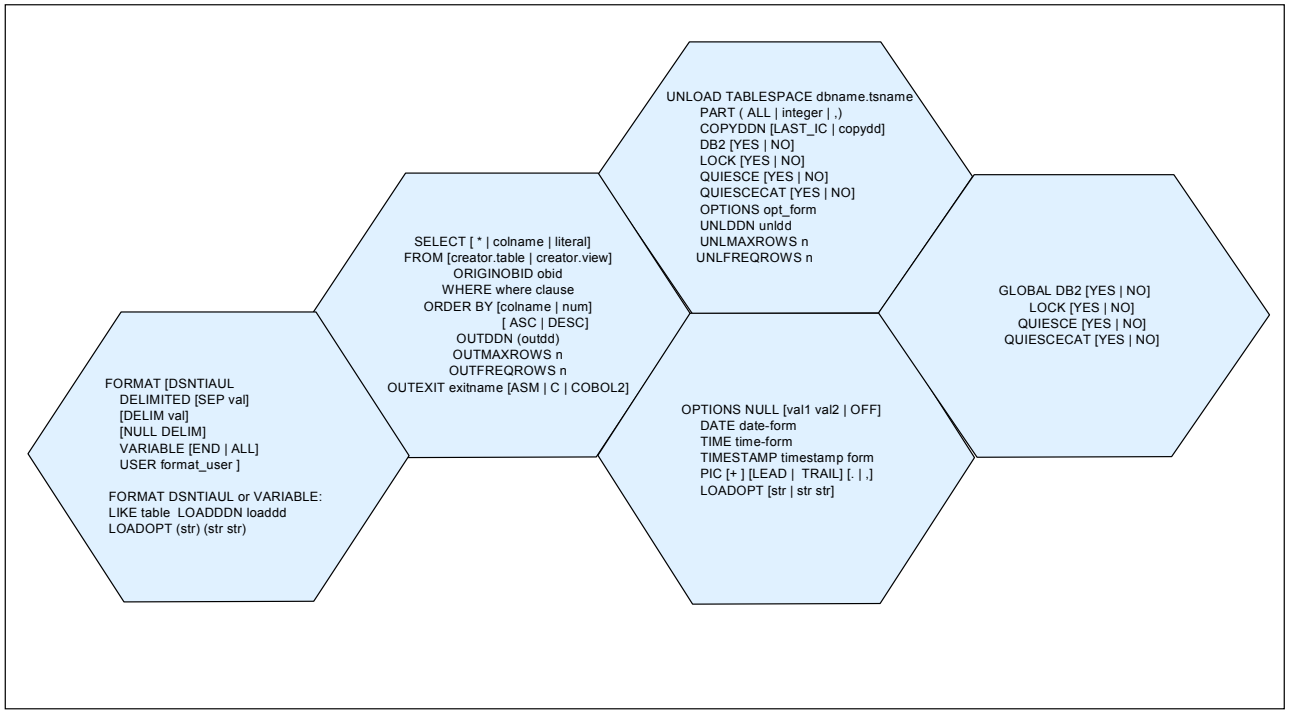

Figure 7-7 Connection diagram of the blocks of HPU

7.5.2 Descriptions of the HPU blocks

The following are descriptions of the HPU blocks:

Unload block

The Unload block Example 7-12 is the main block of the HPU statement. All HPU statements are required to have this block.

Example 7-12 Complete options available in the Unload block

```
UNLOAD TABLESPACE dbname.tsname
    PART ( ALL | integer | ,)
    COPYDDN [LAST_IC | copydd]
    DB2 [YES | NO]
    LOCK [YES | NO]
    QUIESCE [YES | NO]
    QUIESCECAT [YES | NO]
    OPTIONS opt_form
     UNLDDN unldd
     UNLMAXROWS n
     UNLFREQROWS n
```

Options you can specify in the Unload block:

DB2

Specifies the processing to be performed for SELECT statements that are not supported by HPU.

– YES

Indicates that if the SELECT statement is too complex to be handled directly by HPU,DB2 is called to extract the rows.A Warning message (Severity 4)is issued to report this occurrence. The overall return code is raised to 4.

– NO

Indicates that the SELECT statements must be processed by HPU.If a SELECT statement is not supported by HPU, an error is raised and execution stops.Note:the control is done when reading the SYSIN, prior to processing any unload.

– FORCE

Indicates that DB2 must be used to extract the requested rows. This is useful when the SELECT statement uses a filtering predicate that is very efficiently processed through SQL, and the filtering factor is high. That is a small number of rows (say 10%) will be unloaded and an index will be used by DB2. In this case, DB2 processing is faster than HPU's. An Information message (Severity 0) is issued in the report.

DB2 FORCE cannot be used when unloading from an image copy. Attempting to unload from a FIC using DB2 FORCE will result in an error and the execution will stop.

LOCK

Indicates whether or not HPU must lock the table space during the unload:

– YES

The table space is accessed in read-only mode during the execution of HPU.

– NO

The table space is processed without changing its access mode.

QUIESCE

Specifies whether or not to issue a QUIESCE against the table space before unloading it. If the unload is against an image copy, this option is ignored.

– YES

The QUIESCE command is processed if the table space is not in COPY pending status; otherwise, the table space is stopped and restarted.

– NO

The table space is processed without QUIESCE.

QUIESCECAT

Specifies whether or not to issue a QUIESCE on the DB2 catalog's table spaces before unloading.The QUIESCE is done only once, prior to any catalog access, if at least one unload requests it:

– YES

A QUIESCE is to be processed on the catalog tables.

– NO

No QUIESCE is to be processed on the catalog tables.

Global block

In this block you put the global options that you want for all of the output files. The options you specify in the global block will be the default option for all the output files created by the Select statements in SYSIN. However, the options specified in the Unload block takes precedence over the Global block. Hence, these values will be overridden by the options you specify for each column or table in the Unload block.

The options you can specify on the Global block are DB2, LOCK, QUIESCE, and QUIESCECAT, see Example 7-13. Refer to the Unload block for the description of these options.

Example 7-13 Global block

```
GLOBAL DB2 [YES | NO]
    LOCK [YES | NO]
    QUIESCE [YES | NO]
    QUIESCECAT [YES | NO]
```

Select block

In this block, you specify the table, column name or column number of the data that you want to unload. The options available in the Select block is in Example 7-14.

Example 7-14 Select block

```
SELECT [ * | colname | literal]
   FROM [creator.table | creator.view]
   ORIGINOBID obid
   WHERE where clause
   ORDER BY [colname | num]
                       [ ASC | DESC]
   OUTDDN (outdd)
   OUTMAXROWS n
   OUTFREQROWS n
   OUTEXIT exitname [ASM | C | COBOL2]
```

SELECT

The use of the select clause statement in this block is the same as the DB2 SQL Select clause. See *DB2 SQL User's Reference Guide* for further information.

The select statement in this block includes these clauses:

- Where clause
- Order by

ORIGINOBID

This keyword must be specified in conjunction with the COPYDDN statement.It is used when the OBID table in the image copy is not the same as the OBID read in the catalog.This can happen, for example, for an image copy of a table that was dropped and then recreated with a new

OBID

If the source data is an image copy, use this keyword to specify the OBID of the rows to be processed in this image copy.

- integer

If the image copy file contains a *unique* table, the value 0 can be used instead of the OBID of the table. If you specify the value 0, HPU processes the first OBID that is found in the image copy.

- X 'hhhh'

hhhh is the hexadecimal value of the OBID of the table in the image copy.

ORDER CLUSTER

This clause indicates to HPU that the output data set must be sorted according to the index cluster. You must code the ddname (UTPRINT), which contains the sort messages (see “DDNAMES”). If no index is defined on the table, a warning message is issued, and processing continues.

OUTDDN outdd

Specifies the ddname of the sequential output data set containing the unloaded data.You can specify up to 255 ddnames. *outdd* is the base ddname of the output data set. If you want HPU to perform processing in parallel for partitioned table spaces, code in your JCL one outdd *nnn* statement for each partition (outdd *0*1 outdd *0*2 ...outdd *nn*n), where *nnn* is a 2-to 4-digit sequential number that identifies a partition to be unloaded. During the unload process data from each partition is directed to the corresponding ddname. If the corresponding ddname is allocated, it is used for the given partition; otherwise, the base ddname, if allocated, is used. For example:

```
UNLOAD TABLESPACE PART(1,2,4,5)SELECT *FROM Q.T OUTDDN(MYDD)
FORMAT DSNTIAUL
```

If MYDD,MYDD01,and MYDD0004 are allocated, then MYDD contains the rows from partition 2 and 5,MYDD01 contains the rows from partition 1, and MYDD0004 contains the rows from partition 4.

If you do not specify this parameter, then specify UNLDDN on the UNLOAD TABLESPACE statement.

OUTMAXROWS n

Specifies the maximum number of rows, n to be extracted for this SELECT statement. If you are processing a partitioned table space, the number *n* applies to each partition.

OUTFREQROWS n

Specifies the unload sampling frequency. One row every *n* rows will be written to the OUTDDN data set.

OUTEXIT exitname

Specifies the name and the language of the exit that handles the rows during unload processing.The exit must reside in an authorized library, and is loaded dynamically during the execution.The library must be in either the LINKLIST, or in an authorized JOBLIB or STEPLIB. For COBOL/2 and C, the STEPLIB, JOBLIB, or LINKLIST should also point to the LE/370 runtime libraries.

– ASM

  Assembler language

– C

  C language

– COBOL2

  COBOL/2 language

Default: ASM

EBCDIC/ASCII/UNICODE/ASIS

Specifies that the data is unloaded in EBCDIC, ASCII, or UNICODE format using the CCSID of the installation or the specified CCSID.

– ASCII

Indicates that the unloaded data must be in ASCII format.HPU uses the subsystem's ASCII CCSID, unless overridden by specifying the CCSID option.

– ASIS

Indicates that the data is unloaded in its original format.If the specification for the underlying table space cannot be determined (for example, if the data is processed by DB2),the CCSID returned by a standard prepare statement in SQLDA is used.You can also override ASIS by specifying the CCSID keyword.

Specifying ASIS does not mean that no conversion is necessary. Conversion might still be required in some situations, such as between input from SYSIN and the CCSID of the system, or between the CCSID of the system and printed output.

– EBCDIC

Indicates that the data is unloaded in EBCDIC format. HPU uses the subsystem's EBCDIC CCSID, unless you override it by specifying the CCSID keyword.

– UNICODE

Indicates that the data is unloaded in UNICODE format. HPU uses the subsystem's UNICODE CCSID, unless you override it by specifying the CCSID option.

Default: EBCDIC

If the unload format (the unload format is either specified in the SYSIN or in the PARMLIB using the UNLSCHEM variable) is not identical to the system's EBCDIC format, all constants that are specified in SYSIN will be translated to the unload format.

CCSID

Specifies up to three coded character set identifiers (CCSIDs) for the unloaded data.The first specifies the CCSID for SBCS data, the second specifies the CCSID for MIXED DBCS data, and the third specifies the CCSID for DBCS data.If any of these are specified as 0 or omitted, the CCSID of the corresponding data type is assumed to be the same as the installation default CCSID.

CCSID can also be specified at the column level, in the user_block syntax.

Format block

In this block, you specify the format of the output data that will be produced by the HPU statement, see Example 7-15. The Format block can only be attached to a Select block.

Example 7-15 Format block

```
FORMAT [DSNTIAUL
                DELIMITED [SEP val]
                [DELIM val]
                [NULL DELIM]
                VARIABLE [END | ALL]
                USER format_user ]

FORMAT DSNTIAUL or VARIABLE:
  LIKE table  LOADDDN loaddd
  LOADOPT (str) (str str)
```

Format USER options, se Example 7-16.

Example 7-16 Format USER options

```
FORMAT USER:
```

```
(COL [colname | num]
    TYPE type idem scan
    PADDING val
    DELIM val
    LENGTHBYTE [YES | NO]
    LENGTH [REAL | MAX]
    NULLID [YES | NO]
```

DELIMITED delimited_block

Use this keyword to specify that data is to be unloaded in DELIMITED format.For a description of the delimited_block syntax, see “DELIMITED block”.

You have the following three options:

– SEP val

  This keyword defines the separator character to be used in a FORMAT DELIMITED.

  • val can be ’c’ or X ’hh’.

  Default blank

– DELIM val

  val can equal ’c’ or X’hh’.This option defines the delimiter character (there is no default value.)

– NULL DELIM

  This option defines whether the null values should be enclosed by the delimiter.

Char, varchar, graphic, and vargraphic columns are enclosed by the delimiter character.Null columns are not enclosed by the delimiter character if DELIM val and NULL DELIM are coded. All the other types of columns are enclosed by the separator character.

Columns in DATE, TIME, and TIMESTAMP have the format ISO, which means:

– DATE corresponds to format YYYY-MM-DD
– TIME corresponds to HH.MM.SS
– TIMESTAMP corresponds to YYYY-MM-DD-HH.MM.SS.NNNNNN

You can override the default DATE/TIME/TIMESTAMP format by specifying an options_block at the SELECT level.Only a SELECT level options_block will be taken into consideration for this format.

DSNTIAUL dsntiaul_block

Use this keyword to specify that data is to be unloaded in the same format as is produced by the DSNTIAUL program.

USER user_block

Use this keyword to specify that data is to be unloaded as defined by the user. You can specify the format of a specific column using a user_block. Specify FORMAT USER to indicate that you want the unloaded data to be formatted according to the keywords specified in the user_block.You can change field attributes for all selected columns, which means you can specify several keywords for each column, according to the type of data the column contains.

The default values are determined by the values set in the options_block. If all the fields unloaded are fixed, then the default RECFM is FB. If at least one output field is variable, then the default RECFM is VB.

If the LRECL is not specified, HPU determines it at run time using the following rules:

– Fixed: the LRECL of the data set is equal to the sum of the maximum length of fields, regardless of what was specified as the LRECL in the JCL.The output data set will be in FB format.
– Variable and variable blocked: the LRECL of the data set is equal to the sum of the maximum length of fields plus 4,regardless of what was specified as the LRECL in the JCL.The output data set will be in VB format.

The options that you can specify are:

– COL colname /num

Specifies the column's name or number in the SELECT statement.

The attributes that can be specified (in keywords) for each field follow:

– The type of output field
– A padding character for character strings
– A delimiter for fields containing DATE, TIME, or TIMESTAMP
– A choice of whether variable-length fields will make use of their actual length or of the maximum length
– Whether a field is to be preceded by a null indicator
– Whether the two length bytes will be deleted for types of variable-length
– A justification for character or numeric strings.

A description of the keywords that specify these attributes follows:

– TYPE val

The output field format to produce. TYPE can specifies the conversion to be performed.

Example: TYPE CHAR(10)

Default: the default field format for output records is the format specified for columns in the SELECT statement.

– PADDING val

Specifies the padding character *va*l to be used whenever padding is necessary for a column.The padding character can be a single character or a hexadecimal value.If you specify *val* otherwise, only the first character or the first two digits are used.Padding is normally applied to the end of character strings.If JUST RIGHT is specified, the padding is added at the beginning of the string.

The padding is used for the conversion of characters to a string of greater length.

Example: PADDING '*'or PADDING X '00' (binary zero)

A "blank" character is used as the default padding character.

For the DATE, TIME, and TIMESTAMP columns, the default format is the one that is used in the options_block.

Important: When you pad a character field and also do a conversion from EBCDIC to ASCII or vice-versa in one HPU job. The output file will be converted first before it is padded by the value you specify. So you will have a field with EBCDIC characters but with ASCII padding or a field with ASCII characters with EBCIDC padding depending on the conversion you are doing. So it is not recommended that you pad your characters and do the code page conversion at the same time in one HPU statement.

One way to do it is by padding your character fields in your HPU statement. When you already have the output file do the code page conversion through FTP.

– DELIM literal

When specified in a user_block, indicates the delimiter to be used in external DATE or TIME fields.The literal must be one character long (and also one byte long, regardless of the literal CCSID.)

Defaults:

– '-'for DATE fields
– '.'for TIME fields

For the TIMESTAMP column, both delimiters are used.

– LENGTHBYTE

Specifies whether the 2 length bytes for variable-length columns should be written to the output data set:

- YES

  The two length bytes should be written.

- NO

  The two length bytes should not be written.

Default: LENGTHBYTE YES

– LENGTH

Specifies whether the real or maximum length is to be used for fields of variable length.

- REAL

  The length of the field is not to change (value of the two length bytes.)

- MAX

  The output field is to be padded to its maximum length with binary zeros.

Note: This keyword is only useful for variable-length fields.

Default: LENGTH REAL

– NULLID

Specifies whether a null indicator byte is to be created preceding an output field.NULLID can also be specified in the options_block.

- YES

  The null indicator is to be created.This indicator will be set to the value X 'FF' if the column is null; otherwise, the indicator will be set to X'00'.The indicator can be used by LOAD to load null values into a table.The values of the null indicator can be changed with the NULL option.

- NO

  The Null indicator is not to be created.

Default: NULLID NO

– JUST

Specifies whether the output character string is to be justified (aligned).The JUST attribute specifies right or left justification for extended numeric values or for character strings when converting to strings of greater length:

- RIGHT

  Justify the output character string to the right.

- LEFT

  Justify the output character string to the left.

Default: Left justification for conversion between character strings; right justification for numeric conversions in strings.

– NULL

Indicates whether the null indicator is generated in the output data set. NULL can also be specified in the options_block.

- val1 Value of the null indicator when the column value is NULL. val1 can be specified in character (’C’) or hexadecimal (X'hh').
- val2 Value of the null indicator when the column value is NOT NULL. val2 can be specified in character (’C’) or hexadecimal (X'hh').
- OFF No null indicator is generated.

VARIABLE variable_block

Use this keyword to specify that data is to be unloaded in a format that is compatible with the DB2 LOAD data set.

You can specify the following options for formatting the variable columns:

– END

The characteristics and the sequence of fields in the generated data set correspond to those in the SELECT statement. The fields generated in the data set are also typical of those in DSNTIAUL format; the only differences are:

Columns in DATE, TIME, or TIMESTAMP have the format ISO, which means:

- DATE corresponds to format YYYY-MM-DD
- TIME corresponds to HH.MM.SS
- TIMESTAMP corresponds to YYYY-MM-DD-HH.MM.SS.NNNNNN

If a column accepts the nulls, the null indicator is generated in front of the field.This indicator contains the value X ’FF’ if the field is null and X ’00’ if the value is usable.

If the last selected column is variable, the type of the default generated data set will be VB, and this last column will be written only on its effective length; the two length bytes are placed before the column.

You can override the default DATE/TIME/TIMESTAMP format by specifying an options_block at the SELECT level.Only a SELECT level options_block will be taken into consideration for this format.

– ALL

The only difference between this format and the END format is that all the variable columns are written using their actual length.

– LIKE table-name

The use of FORMAT VARIABLE with a table name is the same as for FORMAT DSNTIAUL (described on page “DSNTIAUL block”), and the differences described above are still valid.

LOADDDN loaddd

Use this keyword if you want HPU to create a command data set for the Load utility.

– loaddd

Specifies the name of the DD statement describing the command data set.The corresponding DD statement must be present in the execution JCL.This data set contains the necessary commands for loading a sequential data set using the DB2 LOAD utility.

When the clause LIKE CREATOR-TABLE is not used, the model table is the one referenced in the SELECT statement.

LOADOPT

This keyword enables you to modify the options of the DB2 LOAD command.Use this keyword to specify the options that you want HPU place in the LOAD SYSIN that is created during the unload process.

– tableoptions

Options for the table space

– partoptions

Options for the partition

UNLDDN unldd

Indicates that a physical unload of the table space is to be performed, and specifies the ddname of the output data set.

– unldd

This is the base ddname of the output data set. If you want HPU to perform processing in parallel for partitioned table spaces, code in your JCL one unldd nnn statement for each partition (unldd 01 unldd 02 ...unldd nnn , where nnn is a 2-to 4-digit sequential number that identifies a partition to be unloaded. During the unload process, data from each partition is directed to the corresponding ddname.

If the corresponding ddname is allocated, it is used for the given partition; otherwise, the base ddname, if allocated, is used. For example:

```
UNLOAD TABLESPACE PART(1,2,4,5)UNLDDN(MYDD)FORMAT DSNTIAUL
```

If MYDD,MYDD01,and MYDD0004 are allocated, then MYDD contains the rows from partition 2 and 5,MYDD01 contains the rows from partition 1, and MYDD0004 contains the rows from partition 4.

If you do not specify this parameter, specify OUTDDN on the SELECT statement.

The format of this data set is similar to the format when a DB2 REORG UNLOAD ONLY is done.

UNLMAXROWS n

Specifies the maximum number of rows to unload for a physical unload.If the unload involves a partitioned table space, which is treated partition by partition, then the limit applies to each partition.The variable n must be an integer.

UNLFREQROWS n

Specifies the unload sampling frequency for a physical unload.One row of every n rows will be written to the UNLDDN data set.The variable n must be an integer.

Options block

The options_block enables you to specify the default conversions that will be used with the SELECT statement. This block can be used in the global_block, the unload_block, and the select_block.

Options specified in a global_block or unload_block are only applicable to the USER format, except LOADOPT. The value of the LOADOPT is created by merging values specified in the parmlib, the global_block, the unload_block, and the select_block.However,LOADOPT is also specified in the FORMAT specification, then it is used as is (no merge with previous levels.) Options block, see Example 7-17.

Example 7-17 Options block

```
OPTIONS NULL [val1 val2 | OFF]
DATE date-form
TIME time-form
TIMESTAMP timestamp form
PIC [+ ] [LEAD |  TRAIL] [. | ,]
LOADOPT [str | str str]
```

NULL

Indicates whether the null indicator is generated in the output data set.This keyword has meaning only for the USER format.It also can be specified in the SELECT statement in the FORMAT USER syntax.

- val1 Value of the null indicator when the column value is NULL.
- val1 can be specified in character (’C’) or hexadecimal (X'hh').
- val2 Value of the null indicator when the column value is NOT NULL.
- val2 can be specified in character (’C’) or hexadecimal (X'hh').
- OFF No null indicator is generated.

DATE DATE_x

Specifies the default output format for the DATE columns.Specify DATE_x where x can be any uppercase letter from A through P. The formats for each choice are as follows:

- DATE_A format MM-DD-YYYY
- DATE_B format MM-DD-YY
- DATE_C format YYYY-MM-DD
- DATE_D format YY-MM-DD
- DATE_E format DD-MM-YYYY
- DATE_F format DD-MM-YY
- DATE_G format YYYY-DDD
- DATE_H format YY-DDD
- DATE_I format MMDDYYYY
- DATE_J format MMDDYY
- DATE_K format YYYYMMDD
- DATE_L format YYMMDD
- DATE_M format DDMMYYYY
- DATE_N format DDMMYY
- DATE_O format YYYYDDD
- DATE_P format YYDDD

DATEDELIM literal

Specifies the default delimiter to be used in external date representation. Val must be one character (and one byte long, regardless of the literal CCSID.)

TIME TIME_x

Specifies the default conversion for time representation.Specify TIME_x where x can be any uppercase letter from A through E. The result of each choice is as follows:

- TIME_A format HH.MM.SS
- TIME_B format HH.MM
- TIME_C format HH.MM AM
- TIME_D format HHMMSS
- TIME_E format HHMM

TIMEDELIM literal

Indicates the default delimiter to be used in external time representation. The literal must be 1 character long (and also 1 byte long, regardless of the literal CCSID.)

TIMESTAMP TIMESTAMP_x

Specifies the default conversion for the TIMESTAMP columns.Specify TIMESTAMP TIMESTAMP_x, where x can be an uppercase letter from A through G.The result of each choice is as follows:

- TIMESTAMP_A format YYYY-MM-DD-HH.MM.SS
- TIMESTAMP_B format YYYY-MM-DD-HH.MM.SS.NNNNNN
- TIMESTAMP_C format YYYYMMDDHHMMSS
- TIMESTAMP_D format YYMMDDHHMMSS
- TIMESTAMP_E format YYYYMMDDHHMMSSNNNNNN
- TIMESTAMP_F format YYMMDDHHMMSSNNNNNN
- TIMESTAMP_G format YYYY-MM-DD HH:MM:SS.NNN

For DATE, TIME, and TIMESTAMP columns:

The values DATE_C, TIME_A, and TIMESTAMP_B are used to unload the DATE,TIME, and TIMESTAMP columns, unless other values are specified for the TYPE parameter within the FORMAT USER block

Default values in the OPTIONS block

Variables VUU015/ULDATE,VUU016/ULTIME, or VUU017/ULTMSP at installation or during the customization process.

PIC (parm1, parm2, parm3, parm4)

Defines the numeric data display format used when numeric values are converted for external representation.The PIC keyword has four parameters.The first three must be specified, and the fourth is optional.

The *first parameter* specifies the rules for printing the sign:

- '-' Indicates that the minus sign, '-' ,is present if the number is negative.Otherwise the sign character is a 'blank'.
- '+' Indicates that the sign is always present. Positive values have a '+', and negative values have a '-' sign.
- 'P' Indicates that the padding character is used for positive values and the minus sign,'-',is used for negative values.

The *second parameter* specifies the position the sign relative to the column.

- LEAD

  The sign will be placed in front of the numeric value.(LEAD is ignored for floating point numbers.)

- TRAIL

  The sign will be placed after the numeric value.(TRAIL is ignored for floating point numbers.)

The *third parameter* specifies the decimal separator to be used.

- '.' Use a point as the decimal separator.
- ',' Use a comma as the decimal separator.

The optional *fourth parameter* indicates formatting rules for a nonsignificant zero:

– '*.*' No unnecessary 0 is added to the number either before or after the decimal separator. The number 0 is represented as 0. For example: a DECIMAL(3,4)0.23 is represented as .23; 2 is represented as 2.
– '*.0 ' The number is padded with 0 up to the decimal precision of the column.For example: a DECIMAL(5,3)12.5 is represented as 12.500;0.3 is represented as .300.A DECIMAL(5,0)120 is represented as 120.
– '0.*' No unnecessary number is added to the number after the decimal separator.However,there is always a digit before the decimal point if the precision allows it.For example: a DECIMAL(4,3)1.23 is represented as 1.23;.25 is represented as 0.25;2 is represented as 2.
– '00.*' The integer part of the number, if any, according to the precision and scale of the number, is padded with 0.For example: a DECIMAL(5,2)2 is represented as 002;1.23 is represented as 001.23.
– '0.0' The number is padded with 0 up to the decimal precision of the column, and there is always a digit prior to the decimal point, if the precision allows it.For example DECIMAL(4,3)1.23 is represented as 1.230;0.5 is represented as 0.500.
– '00.0' The integer part of the number, if any, according to the precision and scale of the number, is padded with 0 and the number will be padded with 0 up to the decimal precision of the column.For example:DECIMAL(5,2)0 is represented as 000.00.

Default: The default value is ('-',LEAD,'.','*.*'),meaning that the sign will be printed in front of the numeric value, the sign will only be shown for negative values, the decimal separator will be a point, and no unnecessary 0 is added to the number.

LOADOPT

For this keyword, see the LOADOPT description in the Format block.

LENGTHBYTE

Specifies whether the 2 length bytes for variable-length columns should be written to the output data set.

– YES

  The two length bytes should be written.

– NO

  The two length bytes should not be written.

Default: LENGTHBYTE YES

LENGTH

Specifies whether the real or maximum length is to be used for fields of variable length.

– REAL

  The length of the field is not to change (value of the two length bytes.)

– MAX

  The output field is to be padded to its maximum length with binary zeros.

This keyword is only useful for variable-length fields.

Default: LENGTH REAL

NULLID

Specifies whether a null indicator byte is to be created preceding an output field.This keyword is has meaning only for the USER format.It can also be specified in the SELECT statement in the FORMAT USER syntax.

– YES

The null indicator is to be created.This indicator will be set to the value X ’FF’ if the column is null; otherwise, the indicator will be set to X'00'.The indicator can be used by LOAD to load null values into a table.The values of the null indicator can be changed with the NULL option (see “Options block description”.)

– NO

The Null indicator is not to be created.

NULLPOS

Specifies the position of the NULL indicator.This keyword is has meaning only for the USER format. This option is only available in the OPTION BLOCK.

– BEFORE

Position the null indicator before the data field.

– AFTER

Position the null indicator after the data field.

7.6 Examples on using HPU in batch

Given the numerous options available in an Unload job you can have a lot of choices resulting from the permutation of their selection. This flexibility makes HPU a good tool to move data across different platforms. In this section we show some examples of the flexibility on using HPU.

Example 1: Unloading to an ASCII file with variable format

You can unload a table space to an ASCII file having variable format, see Example 7-18. When a table that resides in DB2 for z/OS is unloaded to an ASCII file then you can move it through FTP to a DB2 residing in a distributed DB2 server.

Example 7-18 Sample JCL to unload to an ASCII file with variable format

```
//PAOLOHPU JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* HP UNLOAD ASCII
//*****************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,DB2UNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//*
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT
      DB2 YES
      QUIESCE YES QUIESCECAT YES
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1) ASCII
         FORMAT VARIABLE ALL
```

```
/*
//SYSPRINT DD  SYSOUT=*
//*
//********* DDNAMES USED BY THE SELECT STATEMENTS **********
//*
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.ASCIIVAR,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
```

Example 2: Unloading to a delimited ASCII file

You can unload to a table space into a delimited ASCII file. The conversion from EBCDIC to ASCII will be done automatically when you specify the ASCII option on the OUTDDN line. In Example 7-19, SANCHEZ1.DEPT table is in the DSNDB04.DEPT tablespace. All columns from the DEPT table will be unloaded in a delimited ASCII file. The file delimiter is a semi-colon ';' and the character delimiter is an asterisk '*'.

Example 7-19 Sample JCL to unload to a delimited ASCII file

```
//HPUNLOAD JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//* HP UNLOAD ASCII
//*************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,DB2UNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//*
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT
      DB2 YES
      QUIESCE YES QUIESCECAT YES
      OPTIONS DATE DATE_A

        SELECT * FROM SANCHEZ1.DEPT
         OUTDDN (UNLDDN2) ASCII
         FORMAT DELIMITED SEP ';' DELIM '*'
/*
//SYSPRINT DD  SYSOUT=*
//*
//********* DDNAMES USED BY THE SELECT STATEMENTS **********
//*
//UNLDDN2  DD  DSN=SANCHEZ1.MD.HPUNLOAD.ASCIIDEL,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
```

Example 3: Unloading non-partitioned table space

You can unload a non-partitioned table space that contains two different tables. The unload can be done in parallel. In Example 7-20 there are two select statements. One for each table. The output of the first select statement is in variable format while the output of the second select statement is in DSNTIAUL format.

Example 7-20 Unloading non-partitioned table space

```
UNLOAD TABLESPACE DB_DB04.TS_DEV1
SELECT * FROM SANCHEZ1.DEPT01
OUTDDN (DDNTBL01)
FORMAT VARIABLE END
SELECT *FROM SANCHEZ1.EMP02
OUTDDN (DDNTBL02)
FORMAT DSNTIAUL
```

Example 4: Unloading all partitions in partitioned table space

To unload all partitions in a partitioned tablespace just do it like you are unloading one non-partitioned table space, see Example 7-21.

Example 7-21 Unloading all partitions in a partitioned table space

```
UNLOAD TABLESPACE DB_DB04.TS_DEV1
SELECT * FROM SANCHEZ1.EMPL01
OUTDDN (DDNTBL01)
FORMAT VARIABLE END
```

Example 5: Unloading selective partitions in a partitioned tablespace

You can selectively unload partitions in a partitioned table space. In this example, we are unloading partitions 1, 2, and 4. You can put the partitions together into one data set. But you also have the option to put them in separate data sets.The output will be in the data set specified by the OUTDD1 DD name specified in the JCL, see Example 7-22.

Example 7-22 Selectively unloading partitions 1,2 and 4 in a partitioned table space

```
UNLOAD TABLESPACE DB_DB04.TS_PROD1 PART (1,2,4)
SELECT * FROM SANCHEZ1.DEPT01
OUTDDN (OUTDD1)
FORMAT VARIABLE END
```

Example 6: Unloading a partitioned table space one file per partition

You can unload a partitioned table space one partition per file, see Example 7-23. To do this, you must code in the OUTDDN statement a generic ddn, and declare this ddn in your JCL using the partition number as a suffix, in our example that is 001, 003 and 004. Remember that in DB2 Version 5, the maximum partition number is 3 digits; therefore, the generic part (the prefix) must not exceed 5 characters.

Example 7-23 Unloading partitions to different files

```
UNLOAD TABLESPACE DSNDB04.TS_PROD2 PART (1,3,4)
SELECT *FROM PAOLOR3.DEPT02
OUTDDN (UNLDD)
FORMAT VARIABLE END

In the JCL, specify the following output DD statements:
//UNLDD001 DD  DSN=PAOLOR2.MD.HPUNLOAD.ASCIIVAR,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
//UNLDD003 DD  DSN=PAOLOR2.MD.HPUNLOAD.ASCIIDEL,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
//UNLDD004 DD  DSN=PAOLOR2.MD.HPUNLOAD.ASCIITIA,
```

```
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
```

Example 7: Unloading to a VARIABLE length file in EBCDIC format

Example 7-24 shows to unload a DB2 table to a variable-length file with EBCDIC format. There will be a consistency point taken during the unload (QUIESCE point.)

Example 7-24 Unloading a DB2 table to a variable-length file

```
UNLOAD TABLESPACE DSNDB04.DEPT
      DB2 YES
      QUIESCE YES QUIESCECAT YES
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1) EBCDIC
         FORMAT VARIABLE ALL
```

Example 8: Using the USER TYPE with data transformation

In Example 7-25 we transform the first column from VARCHAR(10) to CHAR (15). Both columns are left justified. Only the columns specified in the USER statement will have a data type transformation, in our example that is columns 1 and 2. All the other columns in the table will retain their original data type in the unload file.

Example 7-25 Unloading the table to a USER TYPE format

```
UNLOADTABLESPACE DSNDB04.DEPT01

SELECT *FROM SANCHEZ1.DEPT01
OUTDDN (MYDDN)
FORMAT USER (
COL 1 TYPE CHAR(15)
NULLIDYES
JUST LEFT,

COL 2 TYPE CHAR(3)
NULLIDYES
JUST LEFT
)
```

Example 9: Unloading a limited number of rows from a table

This Example 7-26 will unload rows from the DEPT01 table and place it in the data set specified in the MYOUT DD. The number of rows unloaded is 150. This also selects every 10th row and passes it to a user exit named CBLEXIT which is written in COBOL2.

Example 7-26 Unloading limited number of rows and calling a user exit

```
UNLOADTABLESPACE DSNDB04.TS_PROD01
SELECT * FROM SANCHEZ1.DEPT01
OUTDDN (MYOUT)OUTMAXROWS 150
OUTFREQROWS 10
OUTEXIT CBLEXIT COBOL2
FORMAT VARIABLE END
```

Example 10: Unloading from image copies

Example 7-26 shows the image copies of the table space where table PAOLOR2.DEPT is located. ICTYPE **F** and **I** means respectively FULL and INCREMENTAL. The parenthesis with the negative number is not information from the SYSCOPY table, it is included to be used in some of the following examples.

Example 7-27 Image copy information from SYSIBM.SYSCOPY

```
TIMESTAMP                  DSNAME                      ICTYPE
2002-11-14-21.14.29.402332 PAOLOR2.COPY0005.DEPT1ZN9   F     (-1)
2002-11-13-17.12.26.714920 PAOLOR2.COPYI002.DEPT1ZN9   I
2002-11-12-20.11.42.745452 PAOLOR2.COPYI001.DEPT1ZN9   I
2002-11-10-09.10.25.392324 PAOLOR2.COPY0004.DEPT1ZN9   F     (-2)
2002-11-07-10.07.07.039527 PAOLOR2.COPY0003.DEPT1ZN9   F     (-3)
2002-11-03-16.30.28.004300 PAOLOR2.COPY0002.DEPT1ZN9   F     (-4)
```

Full image copies (ICTYPE = F)

The latest copy can be referred in three ways:

- By the key word LAST_IC
- By the negative integer -1
- By data set name

In Example 7-28 you will find syntax for these three alternatives.

Example 7-28 How to refer to the last full image copy

```
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN LAST_IC

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  SYSOUT=*

//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN -1

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  SYSOUT=*

//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN COPYDD

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  DSN=PAOLOR2.COPY0005.DEPT1ZN9,DISP=SHR
```

Older full image copies that the youngest can be referred in two ways:

- Related to the last (-1) with negative integer (-n)
- Data set name

In Example 7-29 you will find syntax to find the third youngest copy for these two alternatives.

Example 7-29 How to refer to an earlier full image copy

```
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN -3

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  SYSOUT=*

//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN COPYDD

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  DSN=PAOLOR2.COPY0004.DEPT1ZN9,DISP=SHR
```

Note: If you are using the negative integer or the LAST_IC key word, HPU always in the job report will verify which data set name was used.

Incremental image copies (ICTYPE = I)

Incremental image copies can only be referred by data set name. See Example 7-30. An incremental copy contains only rows inserted or updated since the last image copy (Full or incremental.)

Example 7-30 How to refer to an incremental image copy

```
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT1ZN9
      COPYDDN COPYDD

       SELECT * FROM PAOLOR2.DEPT
         OUTDDN (UNLDDN1)

//COPYDD   DD  DSN=PAOLOR2.COPYI002.DEPT1ZN9,DISP=SHR
```

7.7 Using the HPU interactively

The HPU can be started interactively. You can use the DB2 Administration tool or the DB2 tools Launchpad to run the tool interactively.

7.7.1 Using the DB2 Administration tool to start HPU

The HPU can be started using the DB2 Administration tool.

Step 1: Launch DB2 Admin

To launch the DB2 Administration tool, go to the ISPF main menu panel. Type ADM in the command line. Your ID needs to have the right authority or privilege to perform administrative tasks on the system. See Figure 7-8.

```
Master Application Menu - SC63
Opt => ADM                                                      Sc => HALF

                                                          USERID - PAOLOR3
Enter SESSION MANAGER Mode ===> NO    (YES or NO)         TIME   - 13:04

  TP  TOPS     - TSO Operator Presentation Sample
  TN  TPNS     - Teleprocessing Network Simulator
  AP  APPC     - APPC Adminstrative Application
  PW  PRINTWAY - IP PrintWay
  IN  INFOPRT  - Infoprint Server
  CV  CICSVR   - CICS VSAM Recovery
  MQ  MQS      - MQSeries
  TE  TERSE    - TERSE/MVS
  VE  VSAMEDIT - VSAM Editor
  FT  NV FTP   - NetView File Transfer Program
  IX  IXFP     - IBM Extended Facilities Product (for RVA)
  FM  FILEMAN  - File Manager
  S2  SDF II   - SDF II Functions
  ADM DB2ADM   - DB2 Administration Tool V4 and Object Compare V2
  AT  DB2AUT   - DB2 Automation Tool

 Use UP and DOWN PF Keys or commands to scroll MENU.
```

Figure 7-8 The ISPF main menu panel

Step 2: Enter the DB2 subsystem name

After entering ADM in the command line you will be presented with another panel with the list of DB2 subsystems active under your MVS. You are required to enter the DB2 subsystem name that you plan to use. Select the DB2 subsystem where the source table spaces are located as shown in Figure 7-9.

```
 DB2 Admin ------------------- Active DB2 Systems ------------------ Row 1 of 3
Command ===>                                                Scroll ===> PAGE


This is a list of the active DB2 systems on this MVS system.

Enter:
DB2 system name ===>

Or select the one you wish to use, or press END to exit.

Sel DB2 System Description                                              Group
--- ---------- ------------------------------------------------------------ -----
s   DB2G
    DB2H
    DB7Y
****************************** Bottom of data ********************************
```

Figure 7-9 The ISPF Active DB2 Systems panel

Step 3: Go to the DB2 system catalog panel

You can move to the DB2 system catalog panel by entering option **1** in the DB2 Administration menu panel as shown in Figure 7-10.

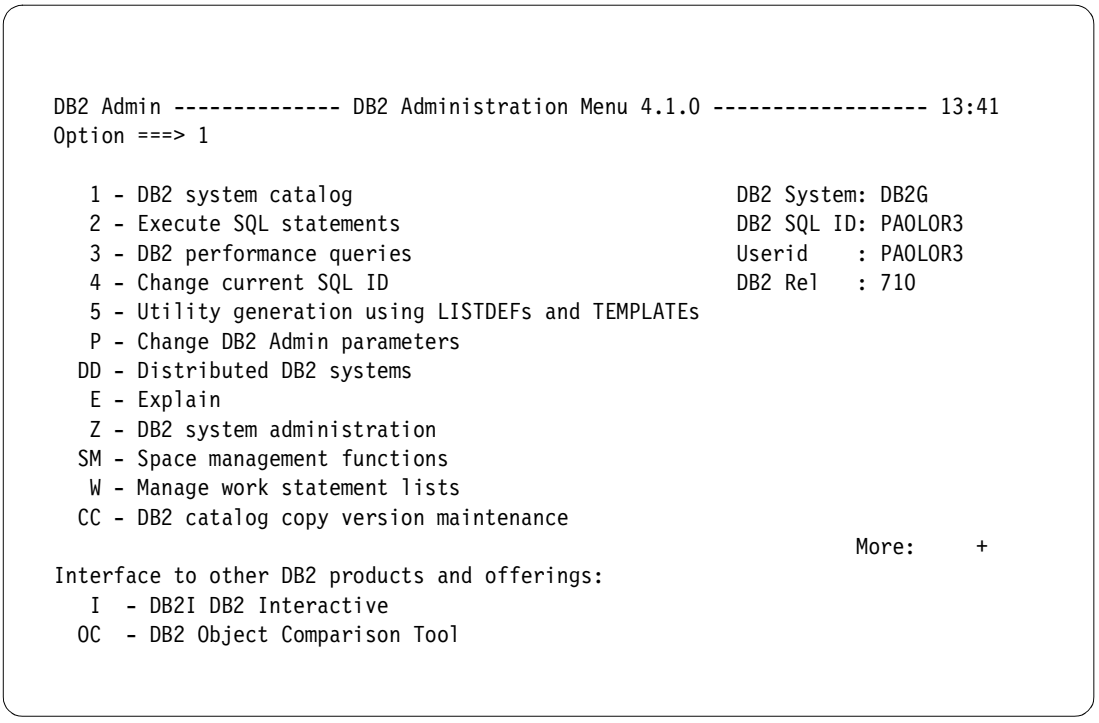

Figure 7-10 The DB2 Administration menu — Option 1

Step 4: Select the table or table space that you want to unload

When you are already in the DB2 Administration system catalog panel enter **T** at the options prompt line. This will bring you to the DB2 Administration tables, views, and aliases panel. In this panel yo can select the table that yo want to unload.

If you want a physical unload of a table space where you unload the entire table space with all the tables in it you should choose the **S** option from the DB2 Administration system catalog panel as shown in Figure 7-11. This will bring you to the DB2 Administration table space panel. In that panel you can select the table space that you want to unload.

```
DB2 Admin ---------------------- DB2G System Catalog ---------------------- 13:53
Option ===> S

                                                                    More:     +
Options:                                                  DB2 System: DB2G
   V - Volumes                                            DB2 SQL ID: PAOLOR3
   G - Storage groups              GA - Authorizations to storage groups
   D - Databases                   DA - Authorizations to databases
   S - Table spaces                SA - Authorizations to tables spaces
   T - Tables, views, and aliases  TA - Authorizations to tables and views
   X - Indexes
   C - Columns                     CA - Authorizations to columns
   Y - Synonyms
   P - Plans                       PA - Authorizations to plans
   K - Packages                    KA - Authorizations to packages
   L - Collections                 LA - Authorizations to collections
   M - DBRMs                       RA - Authorizations to resources
Enter standard selection criteria (An SQL LIKE operator will be used):
Name       ===>                    Grantor  ===>
Owner      ===>                    Grantee  ===>
In D/L/H   ===>                    Switch Catalog Copy  ===> N   (N/S/C)
And/or other selection criteria (option xC shows you columns for option x)
Column     ===>                    Operator ===>       Value ===>
```

Figure 7-11 The DB2 system catalog panel — Option S

On the other hand, you can perform a logical unload by using the DB2 Administration tables, views and aliases menu panel. In this option, you can download tables or views and have an SQL SELECT statement to qualify rows and columns in the unload data. You can enter that panel by selecting the option **T** on the DB2 system catalog panel. See Figure 7-12.

```
DB2 Admin ---------------------- DB2G System Catalog ---------------------- 13:53
Option ===> T

                                                                    More:     +
Options:                                                  DB2 System: DB2G
   V - Volumes                                            DB2 SQL ID: PAOLOR3
   G - Storage groups              GA - Authorizations to storage groups
   D - Databases                   DA - Authorizations to databases
   S - Table spaces                SA - Authorizations to tables spaces
   T - Tables, views, and aliases  TA - Authorizations to tables and views
   X - Indexes
   C - Columns                     CA - Authorizations to columns
   Y - Synonyms
   P - Plans                       PA - Authorizations to plans
   K - Packages                    KA - Authorizations to packages
   L - Collections                 LA - Authorizations to collections
   M - DBRMs                       RA - Authorizations to resources
Enter standard selection criteria (An SQL LIKE operator will be used):
Name       ===>                    Grantor  ===>
Owner      ===>                    Grantee  ===>
In D/L/H   ===>                    Switch Catalog Copy  ===> N   (N/S/C)
And/or other selection criteria (option xC shows you columns for option x)
Column     ===>                    Operator ===>       Value ===>
```

Figure 7-12 The DB2 system catalog panel — Option T

Step 5: Enter HPU line command

Type **HPU** on the line that corresponds to the table that you want to unload. This should be entered on the Sel column inside the DB2 Administration table spaces panel. See Example 7-13.

```
DB2 Admin ----------- DB2G Tables, Views, and Aliases  Row 443 to 455 of 1,000
Command ===>                                              Scroll ===> PAGE

Valid line commands are:
 C - Columns  A - Auth  L - List  X - Indexes  S - Table space  D - Database
 V - Views  T - Tables  P - Plans  Y - Synonyms  SEL - Select prototyping
 ? - Show all line commands

Sel   Name               Owner    T DB Name  TS Name    Cols        Rows Checks
      *                  *        * *        *            *           *      *
----- ------------------ -------- - -------- -------- ------ ----------- ------
      TRY                BARTR2   T DSNDB04  TRY           1          -1      0
      UTLISTE            BART     T DSNDB04  UTLISTE      13          -1      0
      TBPRODUCTS         SC24630O T DSNDB04  TBPRODUC      4          -1      0
      LOGGING2           DB28710  T DSNDB04  LOGGING2      4          -1      0
      LINEITEM           PAOLOR7  T DSNDB04  LINEITEM     16          -1      0
      PLAN_TABLE         USER     T DSNDB04  PLAN1HGJ     51          -1      0
      PLAN_TABLE         BARTR2   T DSNDB04  PLANRTAB     51          -1      0
      EMP                PAOLOR7  T DSNDB04  EMP          14          -1      0
      PHONENO            BART     T DSNDB04  PHONENO       1          -1      0
      PLAN_TABLE         SYS1     T DSNDB04  PLAN1HOV     51          -1      0
HPU   DEPT               PAOLOR7  T DSNDB04  DEPT          5          -1      0
      ACT                PAOLOR7  T DSNDB04  ACT           3          -1      0
```

Figure 7-13 The DB2 Administration tables, views, and aliases panel — Option HPU

You can also enter the **HPU** line command on the DB2 Administration tables spaces panel. Put the command on the Select column corresponding to the table space that you want to unload. See Figure 7-14.

```
DB2 Admin ------------------ DB2G Table Spaces --------- Row 200 to 212 of 407
Command ===>                                              Scroll ===> PAGE

Valid line commands are:
 T -Tables  D - Database  A - Auth  G - Storage group  ICS - Image copy status
 DIS - Display database  STA - Start database  STO - Stop database  X - Indexes
 ? - Show all line commands

Select Name     DB Name   Parts Bpool  L E S I C Tables  Act. pages  Segsz T L
       *        *             * *      * * * * *      *           *      *
------ -------- -------- ------ ------ - - - - - ------ ----------- ------ - -
       CUSTOMER DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
 HPU   DEPT     DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DEPTCOPY DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DEPT1GGC DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DGORDGOD DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DGORDGOP DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DSNRFUNC DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DSNRSTAT DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DSNR1J@X DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DSNR1TUY DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       DSNR1U1H DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
       EMP      DSNDB04       0 BP0    A N A Y Y      1           0      0   Y
```

Figure 7-14 The DB2 Administration table spaces panel — Option HPU

Step 6: Enter the unload options

After entering **HPU** on the line that corresponds to the table, the next options panel that appears is reported in Figure 7-15.

```
HPU -------- DB2G PAOLOR7  . DEPT              - SEL      &ZPREFIX NOT USABLE
Command ===>

Commands :  COLumns    WHERE    ORDERby    OUTddn    FUSER    LOADddn    JCL

Other Option :  PART

SELECT Description ===> PAOLOR7 .DEPT

PART         ===> *     (L : List, * : ALL, nnn : partition number)

FORMAT       ===> 2 ( 1: DSNTIAUL, 2: DELIMITED, 3: VARIABLE, 4: USER)
  DSNTIAUL STRICT  ===>    (Yes, No)
  DELIMITED  SEP        ===> ;   (enter one character or an hexadecimal value)
             DELIM      ===> "   (enter one character or an hexadecimal value)
             NULL DELIM ===> y   (Yes, No)
  VARIABLE    ===>    ( End or All )

  LIKE  creator ===>
        table   ===>
                                                  ( Enter : next screen...)
```

Figure 7-15 Choosing the output format of the unload file

This panel allows you to select the output format. You can select from DSNTIAUL, delimited file, non-delimited file, variable and user format. In our example, we have chosen the delimited format. Our row delimiter is a semicolon(;) and our character delimiter is a double quote (“). The table name chosen here is PAOLOR7.DEPT. Our table will be unloaded in an EBCDIC file format because this is the default value that we specified in the INZTVAR member.

If you choose the DSNTIAUL format it is necessary to fill-up the LOADddn command options. Choosing the DELIMITED format will give you the option to fill-up the SEP and DELIM parameters.

You can choose the file scheme whether EBCDIC to ASCII. You will need this transformation if you will move your data from mainframe to distributed environment. Mainframe computers use EBCDIC characters while machines in distributed platform use ASCII characters.

The CCSID can be specified in the panel shown in Figure 7-16.

```
HPU -------- DB2G PAOLOR7  . DEPT              - SELECT format ------- 17:31
Command ===>

Commands :  COLumns    WHERE    ORDERby    OUTddn    FUSER    LOADddn    JCL

Other Option :  PART

ORIGINOBID   ===>      (HexaDecimal value)
        or   ===>      (Decimal value)

OUTMAXROWS   ===>
OUTFREQROWS  ===>

SCHEME       ===> E ( E: Ebcdic, A: Ascii, S: aSis, U: Unicode)
CCSID SBCS   ===>
      MIXED  ===>
      DBCS   ===>

OUTEXIT  exitname ===>
         in       ===>   ( 1: ASM, 2: C, 3: COBOL2)
                                              ( PF3 : previous screen...)
```

Figure 7-16 Select panel of HPU

The characters used in different languages requires different code pages. For example, if the table you are unloading uses double byte Japanese characters, you need to specify in the DBCS option the appropriate code page number, so that the Japanese characters can be handled properly in the unloaded table. A list of the CCSID codes that you can use is in the appendix.

After selecting the format of the output file, you can select the columns that you want to download by going to the COLUMNS command option box and pressing Enter. You can remove any column by typing **D** or you can arrange the order of the columns by entering the numbers on the Pos in Select column on the panel.

You can also go to the WHERE command option to enter a condition to qualify each row in the select statement of the unload. When you go to this panel you have to type the conditions that will be in the Where clause of your Select statement.

The COLUMNS and the WHERE options are not required to complete the JCL that will be generated by the interactive tool. You can skip these and proceed the OUTDDN command option. The OUTddn is where you enter the output data set that will be used. See Figure 7-18.

Important: The FORMAT and OUTDDN are the only required command options that needs to be filled-up. The other option boxes, like COLUMNS, WHERE, ORDRER BY are not required to be filled-up if you are going to unload the entire table.

```
HPU -------- DB2G PAOLOR7  . DEPT              - SELECT columns  Row 1 from 5
Command ===>                                                 Scroll ===> CSR

Commands :  FORMAT    WHERE    ORDERby    OUTddn    FUSER    LOADddn    JCL

Other Option:  SelectALL


SELECT Description ===> PAOLOR7 .DEPT

Use the following selection codes: U Update, D Delete

  Pos in <-------------------- Columns description -------------------->
S Select Name/Value/Expr.   Trunc L/C/E Type     Length Scale Pos in table
- ------ ------------------ ----- ----- -------- ------ ----- ------------
  .....  ..................         .   ........ ...... ..... <=== NEW
. .....  DEPTNO                     C   CHAR     3              1
. .....  DEPTNAME                   C   VARCHAR  36             2
. .....  MGRNO                      C   CHAR     6              3
. .....  ADMRDEPT                   C   CHAR     3              4
. .....  LOCATION                   C   CHAR     16             5
****************************** Bottom of data ********************************
```

Figure 7-17 Choosing the format of the output of your unload

You do not need to fill-up the Select-columns panel if you are going to unload all the columns from the table.

You can enter the data set name that will be used to contain the unload data as shown in Figure 7-18. This panel is required to be filled-up for all of the command options. After typing the data set name press Enter.

```
HPU --------- DB2G PAOLOR7  . DEPT              - OUTDDN list -------- 18:15
Command ===>                                                 Scroll ===> CSR

Commands :  FORMAT    COLumns    WHERE    ORDERby    FUSER     LOADddn    JCL


SELECT Description ===> PAOLOR7 .DEPT


Use the following selection codes:
S Select a file, D Delete a file, C Copy a file to a new file.

S Data Set Name                                Disp
- ---------------------------------------- -------------
  PAOLOR3.MD.HPUNLOAD.ASCDATA1.....         <=== NEW
****************************** Bottom of data ********************************
```

Figure 7-18 The HPU OUTddn command panel

In Figure 7-19, we are using a data set that does not yet exist in the DASD with DISP=(NEW,CATLG,DELETE).

```
HPU --------- DB2G PAOLOR7  . DEPT              - OUTDDDN file ------- 18:38
Command ===>


                    Dataset with formated result of SELECT

Data Set Name          : PAOLOR3.MD.HPUNLOAD.ASCDATA1

Disp                ===> ( N , C , D )  (New/Old/Mod) (Del/Keep/Catlg)
Generic unit        ===> SYSDA
Volume serial       ===>        -        -        -        -
FORMAT              ===> FB  (F, FB, V, VB)
LRECL               ===> 130
BLKSIZE             ===> 26000
Primary quantity    ===> 10
Secondary quantity  ===> 20
Space units         ===> TRKS   (BLKS, TRKS, CYLS)

for tape unit
Label               ===>
```

Figure 7-19 Panel for output data set attributes

Step 7: Review the JCL

After filling-up the required data set attributes, press F3. You will go back to the OUTddn panel (Figure 7-18 on page 185). Review the JCL generated by putting the cursor on the JCL command option box and press enter. Choose EDIT in the listbox and press Enter. See Figure 7-20. The JCL that was generated will appear on the screen, and you can edit to see if there are some parameters that needs to be changed.

```
HPU --------- DB2G DSNDB04  . DEPT     - SELECT statement list Row 1 to 1 of 1
Command ===>                                                Scroll ===> CSR

Commands :  PART    COPyddn    OPTions   UNLddn    JCL
                                                 EssssssssssssN
                                                 e   EDIT      e
                                                 e   SUBmit    e
                                                 e   SaveJCL   e
New Select statement ? :   (Yes)                 DsssssssssssM
    Interactive Select :   (Y/N, default is Yes)
 or
Use the following selection codes:
S select statement, D delete statement, C Copy statement to new statement

S Select description            Inter. Creator  Table or view
- ---------------------------- ------ -------- ------------------
. PAOLOR7 .DEPT                  Y    PAOLOR7  DEPT
****************************** Bottom of data ********************************
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  Menu  RefList  RefMode  Utilities  Help
```

Figure 7-20 Viewing the JCL

After you have reviewed the JCL, as listed in Figure 7-21, you can submit the job on that panel by typing SUB or SUBMIT on the command line. Or you can do it through the HPU interactive panels. Put the cursor to the JCL command option box. Press the Enter key and a listbox will appear. Then Position your cursor on the SUBMIT line then press Enter. This will submit the JCL generated by the HPU tool for CPU for processing.

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       PAOLOR3.SC63.SPFTEMP1.CNTL                    Columns 00001 00072
Command ===>                                               Scroll ===> PAGE
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 //PAOLOHPU  JOB PAOLOR21,'DB2 UNLOAD',
000002 //          MSGCLASS=A,CLASS=A,NOTIFY=&SYSUID,
000003 //          REGION=OM
000004 //*
000005 //*
000006 //*****************************************************************
000007 //*                    H P U   P R O D U C T                      *
000008 //*****************************************************************
000009 //*                                                               *
000010 //* UNLOAD GENERATED BY PAOLOR3   ON   02/10/30  19:13            *
000011 //*                                                               *
000012 //*****************************************************************
000013 //UNLOAD   EXEC PGM=INZUTILB,PARM='DB2G,HPU'
000014 //STEPLIB  DD DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
000015 //         DD DSN=DB2V710G.SDSNEXIT,DISP=SHR
000016 //         DD DSN=DB2G7.SDSNLOAD,DISP=SHR
```

Figure 7-21 Display of JCL and final revisions

The JCL generated by the HPU interactive tool is in Example 7-31. This job will produce a delimited file in EBCDIC file format. All of the contents of the DEPT table will be unloaded to PAOLOR3.MD.UNLOAD.ASCDATA1 data set.

Example 7-31 JCL generated by the HPU interactive tool

```
//PAOLOHPU  JOB PAOLOR21,'DB2 UNLOAD',
//          MSGCLASS=A,CLASS=A,NOTIFY=&SYSUID,
//          REGION=OM
//*
//*
//*****************************************************************
//*                    H P U   P R O D U C T                      *
//*****************************************************************
//*                                                               *
//* UNLOAD GENERATED BY PAOLOR3   ON   02/11/01  13:28            *
//*                                                               *
//*****************************************************************
//UNLOAD   EXEC PGM=INZUTILB,PARM='DB2G,HPU'
//STEPLIB  DD DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD DSN=DB2G7.SDSNLOAD,DISP=SHR
//O11      DD DSN=PAOLOR3.MD.HPUNLOAD.ASCDATA1,
//O11      DD DSN=PAOLOR3.MD.HPUNLOAD.ASCDATA1,
//            DISP=(NEW,CATLG,DELETE),
```

```
//            UNIT=SYSDA,
//            SPACE=(TRK,(10,20)),
//            DCB=(RECFM=FB,LRECL=130,BLKSIZE=2600)
//UTPRINT  DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
UNLOAD TABLESPACE DSNDB04.DEPT

SELECT  *
 FROM
PAOLOR7 . DEPT
OUTDDN ( 011 )
FORMAT DELIMITED
  SEP ';'
  DELIM '"'
  NULL DELIM
/*
```

Step 8: Submit the job

After reviewing the JCL you can submit the job by going back to the command options panel by pressing F3. Then put the cursor on the JCL command option box and press Enter. The listbox will appear again, but this time choose SUBMIT. The job will run like a regular JCL job. Check the outlist on the SDSF panel.

Successful execution of the unload will result in return code equal to 0 (RC=0). See Figure 7-23 for a sample outlist of a successful unload. A return code of 4 means there are warnings on the unload operation. Look for the warning messages in the outlist. If it the warning is tolerable then you can consider the unload successful. Ask your system programmer about the warning messages. Any return code greater than 4 (RC=4) is considered unsuccessful.

```
Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF OUTPUT DISPLAY PAOLOHPU JOB10924  DSID     2 LINE 0       COLUMNS 01- 80
COMMAND INPUT ===>                                           SCROLL ===> CSR
******************************** TOP OF DATA ********************************
                  J E S 2  J O B  L O G  --  S Y S T E M  S C 6 3  --  N O D E

19.22.37 JOB10924 ---- WEDNESDAY, 30 OCT 2002 ----
19.22.37 JOB10924  IRR010I  USERID PAOLOR3  IS ASSIGNED TO THIS JOB.
19.22.37 JOB10924  ICH70001I PAOLOR3  LAST ACCESS AT 19:21:47 ON WEDNESDAY, OCT
19.22.37 JOB10924  $HASP373 PAOLOHPU STARTED - INIT 1    - CLASS A - SYS SC63
19.22.37 JOB10924  IEF403I PAOLOHPU - STARTED - ASID=03EA - SC63
19.22.38 JOB10924  IGD01008I &DSN    = SYS02303.T192238.RA000.PAOLOHPU.R0102065
19.22.38 JOB10924  IGD01008I &UNIT   = SYSDA
19.22.38 JOB10924  IGD01008I &ALLVOL =
19.22.38 JOB10924  IGD01008I &ANYVOL =
19.22.39 JOB10924  INZX006 DEPT     TABLESPACE UNLOAD PHASE STARTED
19.22.40 JOB10924  INZX090 011      : 43 RECORDS WRITTEN
19.22.40 JOB10924  -                                        --TIMINGS (MINS.)-
19.22.40 JOB10924  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOC
19.22.40 JOB10924  -PAOLOHPU          UNLOAD      00    428    .00    .00    .0
19.22.40 JOB10924  IEF404I PAOLOHPU - ENDED - ASID=03EA - SC63
```

Figure 7-22 SDSF panel — Unload execution results

You can view the summary of the tablespace unload at the bottom of the outlist as listed in Figure 7-23. It will tell you how many rows were unloaded, and how many were discarded, the number of pages read and the number of pages in error.

```
TABLESPACE UNLOAD STATISTICS          UT4100 DB2 HIGH PERFORMANCE
* TABLE              * PART NO. * ROWS READ  * ROWS KEPT  * IX SCAN  *
*                    *          *            *             *          *
* DEPT               *        0 *         43 *         43 *    0 %   *
* INVALID ROWS .................*          0 *          0 *          *
* TABLESPACE TOTAL .............*         43 *         43 *    0 %   *
NUMBER OF PAGES READ ...      12
NUMBER OF PAGES IN ERROR       0
****************************** BOTTOM OF DATA ********************************
```

Figure 7-23 Outlist of HPU on the SDSF panel.

You can exit the HPU panel by pressing F3 several times until you reach the ISPF panels. You can view the output file on the Data set list utility panel (P.3.4) of ISPF. In Figure 7-24 the example is for a delimited EBCDIC file.

```
HPU --------- DB2G PAOLOR7  . DEPT              - OUTDDN list -- Row 1 from 1
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       PAOLOR3.MD.HPUNLOAD.ASCDATA1                   Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 "A00";"SPIFFY COMPUTER SERVICE DIV.";"000010";"A00";"                "
000002 "A01";"SPIFFY COMPUTER SERVICE DIV.";"000011";"Z01";"                "
000003 "A02";"SPIFFY COMPUTER SERVICE DIV.";"000012";"A02";"                "
000004 "B00";"PLANNING";"000020";"A00";"                "
000005 "B01";"PLANNING";"000021";"Z01";"                "
000006 "B02";"PLANNING";"000022";"A02";"                "
000007 "C00";"INFORMATION CENTER";"000030";"A00";"                "
000008 "C01";"INFORMATION CENTER";"000031";"Z01";"                "
000009 "C02";"INFORMATION CENTER";"000032";"A02";"                "
000010 "D00";"DEVELOPMENT CENTER";;"A00";"                "
000011 "D01";"DEVELOPMENT CENTER";;"Z01";"                "
000012 "D02";"DEVELOPMENT CENTER";;"A02";"                "
000013 "D10";"MANUFACTURING SYSTEMS";"000060";"D55";"                "
000014 "D11";"MANUFACTURING SYSTEMS";"000060";"D55";"                "
```

Figure 7-24 Output data set of HPU

If you will move the data to a DB2 table in a distributed platform (UNIX, Windows, Linux) you need to transform your file to ASCII format. But if you plan to move your data to another mainframe DB2 table then you can leave it as an EBCDIC file.

7.8 HPU performance measurements

The new Version 2.1 of HPU performs better in CPU, when compared to DB2 for z/OS Version 7 Unload, except in cases where small number of rows fetched are returned (case 7 and 9). HPU 2.1 is also showing comparable elapsed times with DB2 Unload (5% difference.)

The test cases are the following:

```
Case 1:   UNLOAD FROM TABLESPACE
Case 2:   UNLOAD FROM IMAGE COPY
case 3:   UNLOAD SELECT * FROM TABLE
Case 4:   UNLOAD SELECT COL A,COL B,COL C,COL D,COL E
Case 5:    UNLOAD SELECT *
             WHERE  COL C <> '061'     >>>>>>>>>>>>>>> non-indexed column
Case 6:   UNLOAD SELECT COLA,COLB,COLC,COLD,COLE
             WHERE  COL C <> '061'
Case 7:   UNLOAD SELECT * FROM TABLE
             WHERE  COL C = '061'
Case 8:   UNLOAD SELECT * FROM TABLE
             WHERE  COL D <> 9          >>>>>>>>>>>>>>> indexed column
Case 9:   UNLOAD SELECT * FROM TABLE
             WHERE  COL D = 9
```

The data definition and type of processing are:

- Table has 20 million rows, 20 partitions, and 588,320 pages
- Case 5 and 6 return 19,649,999 records
- Case 7 returns 350,001 records
- Case 8 returns 20,000,000 records
- Case 9 returns 0 record
- Output from case 2:

```
                        HPU          DB2 UNLOAD
========================================
Record format           VB           VB
Record length           121          121
Block size              27998        27998
Allocated cysl/extents2 887/62       790/5
Used cyls/extents2       887/62      790/5
```

Both sets of measurements were preceded by RUNSTATS execution and were run on a G7 zSeries engine with z/OS 1.3.

Table 7-13 shows the CPU and elapsed times of the executions of HPU and Unload for a table with one index. From these results we can observe that:

- There is no significant difference in elapsed time of HPU2.1 and DB2 V7 Unload.
- HPU2.1 takes two to four times less CPU time when most rows in a table are to be unloaded, with or without predicates.
- HPU2.1 takes 20% more CPU time when the number of rows to be unloaded is small with good filtering predicates.

Table 7-13 HPU vs. Unload — Table with 1 index

Table with 1 IX	HPU V2R1 (time in seconds)		Unload (time in seconds)	
	CPU	Elapsed	CPU	Elapsed
Case # 1	23.6	183	56	192
Case # 2	19.5	228	79	237

Table with 1 IX	HPU V2R1 (time in seconds)		Unload (time in seconds)	
	CPU	Elapsed	CPU	Elapsed
Case # 3	26.2	184	NA	NA
Case # 4	21.8	184	73	175
Case # 5	34.5	184	65	189
Case #6	30.3	184	85	175
Case # 7	25.1	184	21	174
Case # 8	34.3	184	67	191
Case # 9	24.7	184	21	175

Other measurements have shown that there is no difference as more indexes are added.

7.9 Considerations

The HPU for z/OS was created to provide an alternative to unload data from DB2 for z/OS tables. The file format options in the output data allows it to be loaded to other tables in the DB2 for z/OS database or across distributed platforms, using the ASCII and DELIMITED format. The main difference of HPU is that it performs a tablespace unload by directly accessing the data from the VSAM clusters using the VSAM buffering capability. Even though HPU can execute the unload directly from the VSAM clusters, it still needs DB2 to be up and running, in order to access the catalog and resolve all the object definitions.

The parallel execution of the select statements makes efficient use of each I/O operation. Each row retrieved is offered to all select statements in the HPU job. It does not have to go back to the same row in the table even if a different Select statement requires this row.

Performance tests on the HPU shows comparable performance to DB2 Unload utility in terms of elapsed time. HPU takes less CPU time in execution, and also provides you with more versatility because you can use the full SQL select statement to select the rows and columns that you want to be unloaded.

HPU can directly process all types of SQL statements, but any SQL which cannot be resolved by processing the data with just one pass, will be given back to DB2 for execution. For example, HPU cannot directly perform a table JOIN or any column function. Hence, if at all possible, you could try to simplify the SQL statement that you use in the HPU statement, so that it can be performed by HPU without calling the DB2 engine. This will save you CPU time.

Whether or not HPU uses DB2 to access the rows from the table is transparent to the user if the option DB2 has been set to `YES`.

You can also force the use of DB2 using the FORCE parameter when you know that the result set is very small, and DB2 can take advantage of the filtering effect.

It is also highly recommended to specify QUIESCE YES and QUIESCECAT YES. This will create a consistency point in the database and make sure that structure changes (such as those reflecting table views used for unloading) are implemented. The buffer pools will be flushed and data will be written on disk. If this option is not specified, you could have some updates on your table that will not be included in the output data. A new option can eliminate the need for QUIESCECAT by reading the DB2 catalog with SQL rather than the VSAM structures.

If you are handling large object binary (LOB) data in your table, you should use DB2 Cross Loader to perform the unload and load of data. HPU does not support LOB SQL data type. See the redbook *Large Objects with DB2 for z/OS and OS/390,* SG24-6571 for details on functionalities and restrictions.

The output file of the HPU can only be moved across different platforms through FTP or other file transfers methods. You cannot take advantage of a Federated Database set-up where you can move data from one table to another by just using nicknames. Moving data through HPU is always a three step process:

1. unload data from source table,
2. FTP to the other machine,
3. load the data to the target table.

We have discussed the advantages and disadvantages of this tool. It is really up to the database personnel to decide on its applicability and the related trade-offs.

8

IBM DB2 High Performance Unload for Multiplatforms

This chapter provides a description of the IBM DB2 High Performance Unload (abbreviated to HPU or HPU for MP in this chapter to differentiate it from the HP for z/OS) tool for distributed environments.

The topics we discuss in this chapter are:

- General overview of HPU features
- Using HPU for MP
- HPU examples
- Performance measurements
- Comparison with the Export utility

8.1 An overview of HPU for Multiplatforms

IBM DB2 High Performance Unload (HPU) for Multiplatforms (MP) is a tool not included in the DB2 UDB product distribution. This product has to be purchased separately and installed on all DB2 server nodes. The installation procedure is outlined in the *DB2 High Performance Unload for Multiplatforms Version 2 Release 1 User's Guide*, SC27-1623-01. More details are currently included in the Readme file distributed with the product. The latest revision level will always be reflected at the Web site:

`http://www-3.ibm.com/software/data/db2imstools/`

The current version of HPU for MP is Version 2.1. It officially works with DB2 UDB Version 7.1 and above, though HPU tool needs FixPak 3 to work with DB2 UDB V8. This maintenance level, equivalent to V1R2M3 was made available while this redbook was being written, hence some of our results and recommendations are also applicable to DB2 UDB V8. During our project we worked with both levels at modification 2 and 3. Informal tests did show that it also works with DB2 UDB for UNIX V6.

HPU for MP can increase performance by circumventing the database manager. Instead of accessing the database by issuing SQL commands against the DB2 database manager, as typical database applications do, HPU itself translates the input SQL statement and directly accesses the database object files. An unload from a backup image may be performed even if the DB2 database manager is not running. Active DB2 database manager is needed to verify that a user not belonging to the sysadm group does have authority needed to run the HPU tool.

HPU can unload data to flat files, pipes, and tape devices. Delimited ASCII and IXF file formats are supported. The user format option is intended to be used to create a file format compatible with the positional ASCII (ASC) formats used by the other DB2 tools and utilities. Creating multiple target files (location and maximum size can be specified) allows for better file system management.

A partitioned database environment HPU with FixPak 3, offers the following features:

- Data from all partitions can be unloaded to multiple target files.

  The syntax allows you to unload, with a single command, on the machine where the partition is, or to bring everything back to the machine you are launching HPU from. The command **`OUTPUT(ON REMOTE HOST "/home/me/myfile")`** creates a file per partition on the machine where the partition reside. Of course the path /home/me/ must exist on each machine impacted by the unload.

- A partitioned table can be unloaded into a single file.

  The command **`OUTPUT(ON CURRENT HOST "/home/me/myfile")`** creates only the file **`myfile`** on the machine you are running from, and will contain all the data of the unload. This is the default, for compatibility reasons, while multiple files will offer better performance.

  Note: The new option **`ON "mynamedhost" HOST`** behaves like **`ON CURRENT HOST`**, except that the output file will be created on the specified host rather than the current host. A restriction exists that the named host must be part of the UDB nodes.

- A subset of table nodes can be unloaded by specifying command line options or through a control file or both.

  The OUTPUT command now supports the FOR PARTS() clauses. The appropriate combination of these clauses allows you the needed flexibility.

HPU tool is an executable externally to DB2 UDB. Input parameters are specified either as command line options or through a control file. HPU can also be defined as a Control Center plug-in.

For detailed information about the HPU, command line syntax, and control file syntax, please consult *IBM DB2 High Performance Unload for Multiplatforms User's Guide*.

8.2 Installing and configuring HPU for MP

In this section we explain how to install and configure HPU for MP.

8.2.1 System requirements

The following table lists the operating systems that HPU supports, as well as the approximate amount of memory and disk space required to use HPU on each system.

Table 8-1 Memory and disk requirements

Operating system	Memory	Disk space
IBM AIX ®4.3 and up	32 MB RAM	42 MB
Linux IA-32	32 MB RAM	42 MB
Hewlett-Packard HP-UX 11.0	32 MB RAM	34 MB
Sun Solaris 2.6,7,and 8	32 MB RAM	32 MB
Windows 2000 and Windows NT 4.0	32 MB RAM	31 MB

8.2.2 Installation considerations and prerequisites

Before you begin installing HPU, review the following information:

- You must have an IBM DB2 UDB Version 7.1 or above in order to use this version of HPU.
- You must install HPU on the same computer where the DB2 tables that you want to unload are physically located.
- You must have the following information:
  - The name and version of your operating system
  - The root password for UNIX or Linux (you must be super-user root)
  - If you are installing on a Windows host, the administrative permissions associated with your user ID
  - If you have a permanent license for HPU, your specific nodelock file authorization

Important: The install setup will fail if the system default permissions are less than 777 and the root super-user mask is different than 022.

All installed directories must have .drwxr-xr-x .permissions.

8.2.3 Installing HPU for MP

The following sections provide instructions for installing HPU on Windows, Linux, and UNIX systems.

Starting the installation process on Windows

On Windows systems, place the CD into the CD-ROM device and the setup will start automatically.

If the automatic start does not work, from File Manager or Windows Explorer, locate your CD-ROM drive and then double-click the **setup.exe** file.

Note: The Windows version must be installed by the Install Administrator only. The installation is then available to all users with administration rights belonging to an administrator group.

On NT 4.0,the hpuplug.zip file does not get copied into the sqllib \cc directory and renamed to db2plug.zip. To work around this problem:

1. Copy the hpuplug.zip file from x:\Program Files \IBM \hpu \V2.1 \java and move it to x:\Program Files \cc.
2. Rename hpuplug.zip to db2plug.zip.

If you are using the Control Center plug-in:

If your HPU installation suddenly disappears, it is possible that the installation of another product into the Control Center has overlaid the HPU plug-in. To fix this problem, you must re-install the HPU plug-in.

To reinstall the HPU plug-in:

1. Run the SETUP command.
2. Select **CUSTOM** install.
3. Click the **HPU** button.
4. Follow the instructions that appear on each panel.

Installing HPU on UNIX and Linux systems

To install HPU on UNIX systems:

1. Insert the product CD into the CD-ROM drive on the UNIX computer where you are installing HPU.
2. Log on to the host computer with the user ID root.
3. Mount the CD on a suitable mount point (for example, /mnt/cdrom).
4. Verify that the DISPLAY environment variable is correctly set.
5. Check the Xserver authorizations (you must be able to start a xterm window.)
6. Change to the directory corresponding to the platform under the CD mount point. For example:
   - For AIX systems:cd /mnt/cdrom/aix
   - For UNIX systems:cd /mnt/cdrom/linux
7. From the command line, type the command **`./setupxxx`**, where xxx is the platform. Examples are:

   ```
   ./setupaix or ./setupsun
   ```

The initial installation screen is displayed.

Note: In case of a multi-node environment, HPU must be installed on all physical nodes. You can only execute HPU on the catalog node.

The HPU installation wizard

This section describes the installation wizard for installing HPU on Windows, Linux, and UNIX.

Welcome page

The first page is the Welcome page. Click **Next** to continue the installation.

Note: At any time during the installation process, you can click **Cancel** to stop the installation program or **Back** to return to the previous page.

Choose Destination Location page

The Choose Destination Location page of the Installation wizard allows you to specify a default destination folder for HPU or to change a destination folder.

Choose from the following options to specify the installation directory:

1. Click **Next** to set HPU as the default destination folder (where the product is installed.)
2. Click **Browse** to locate a different installation directory.

The default installation directories are listed below.

Table 8-2 Default installation directories

System	Installation directory	Temporary directory
UNIX and Linux	/opt/IBM/hpu/V2.1 /	tmp
Windows	C:\Program Files \IBM \hpu \V2.1	C:\Temp

User Input Parameters page

In the User Input Parameters page of the Installation wizard, enter the required information:

- The default instance name. This name must correspond to an existing DB2 instance on the host on which you are installing HPU.
- The default database name. This database must exist within the default instance. The default database name and the default instance name will be used by HPU when a specific instance or database is not specified in the command line or the control file.
- The DB2 directory as the instance home of the default instance. A sqllib directory must exist under the DB2 directory.

After the required input values are correct, click **Next**.

Select Components page

Choose from the following options to specify the installation type for Select Components:

- To install HPU unload and network services (Typical installation), click **Next**. This option will skip the Select Custom Components page.
- To select specific HPU features that you want to install, click **Custom**, then click **Next**. The Select Custom Components page is displayed.

Select Custom Components page

To customize your installation, choose from the following HPU components:

- HPU unload feature

  Selecting this option specifies that you want to install the program db2hpu. This module is used to unload the local DB2 files. This option must be used on a system where a DB2 server is installed.

- HPU network feature

  Selecting this option specifies that you want to install the HPU network server. The server is a specific service that allows HPU to unload DB2 files through the network. This option must be installed on the server if you want to use the GUI to communicate with it, or if this is a partitioned (EEE) environment.

Resume Before Install page

Before you begin the installation, review your selections and then choose one of the following options from the Resume Before Install page:

- Click **Back** to return to a previous page to change any of your selections.
- Click **Next** to install the product as it is shown in the summary. When you click **Next**, a page showing the progress of the installation is displayed.

The installation program installs HPU into various directories. For more information on these directories, refer to page 11 of the *IBM DB2 High Performance Unload for Multiplatforms User Guide*, SC27-1623-01.

Using a response file to install HPU

On a system where you cannot use the graphic wizard to install HPU, you can run the setup module with a silent option.T do this, you must create an ASCII text response file with all of the required information:

1. Create a response file.

   A typical response file is shown in the following example:

   ```
   #specify silent install
   -silent
   #Main product directory
   -P hpu.installLocation="/home/IBMhpu/V2.1"
   #Default instance name
   -W userInput.defaultInstance="db2instance"
   #Default database name
   -W userInput.defaultDB="DATABASE1"
   #Default instance home directory
   -W userInput.defaultHome="/db2home/db2instance"
   ```

   To add comments, precede the comment line with a pound sign (#).

   The keyword –silent specifies to install in silent mode, where the installation is performed with no user interaction.

   Main product directory corresponds to the value of the Destination Location page in the Install wizard.

   The other parameters, default instance namef, default database name, and default instance home directory correspond to the Input fields of the User Input Parameters page from the Install wizard.

2. Start the silent install.

To start the silent install, you must add –options response_filename as a parameter of the **./setup** command.

For example:

```
/mnt/cdrom/aix/setup -options /root/db2hpu/responsefile
```

To use a response file, specify the -options parameter on the command line, followed by the absolute path to the response file. An absolute path to the response file on a Windows system might look like this:

```
c:\directory \responsefile
```

while on a UNIX system, the absolute path might look like this:

```
/root/db2hpu/responsefile
```

8.2.4 Installation directories and files

Files and directories necessary for the installation of HPU are listed in the following section.

Installation directories

The HPU installation program creates the following default product directory during the installation (unless you specify a different directory):

On UNIX and Linux:

```
/opt/IBM/kk/V2.1
```

On Windows:

```
\Program Files \IBM \hpu \V2.1
```

The following table lists the directories and significant files installed in your product directory for all supported systems.

Table 8-3 Directory description

Directory	Description
bin	Contains the executables for HPU:db2hpu,db2hpudm (extensions appear as .exe on Windows systems.)
lib	Contains the dynamic libraries (*.so on UNIX r *.dll on Windows.)
cfg	Contains the configuration file for HPU defaults:db2hpu.cfg
msg	Contains language-dependant messages and resource files.
java	Contains all files (classes, jar files, zip files,...) needed by the .HPU plug-in for Control Center feature.
_jvm	Contains all of the Java Machine files that are used by the install and uninstall setup processes.
_uninst	Contains information installation files.These files will be used by the automatic uninstall setup.

Directory	Description
sample	Contains sample control files for HPU that you can run against the DB2 SAMPLE database. Refer to Appendix A, of the *HPU for Multiplatforms User Guide*, SC27-1623-01
help	Contains the help files.
install	Contains the files used by the installation process.
images	Contains the images associated with message

After installing HPU, you must modify the config file.

HPU configuration file

The db2hpu.cfg file drives all default values. This file is an ASCII text file, and it is located in the cough directory.

A typical db2hpu.cfg file is shown in the following example obtained with FixPak 2 level.

Example 8-1 db2hpu.cfg with default values

```
#HPU default configuration
   version=02.01.000
   pagesizes=4:8:16:32
   bufsize=8388608
   memsize=8388608
   db2dbdft=SAMPLE
   db2instance=db2inst1
   maxunloads=2
   maxsockets=4
   insthomes=db2inst1:/home/db2inst1
```

Lines beginning with a pound symbol (.#.) and empty lines are ignored. Each parameter is defined on one line and that format is: keyword=value. There is no space between keyword and the equal sign (.=.). There is also no space between the equal sign (.=.) and value.

pagesize

This parameter defines the size of the DB2 pages that will be unloaded. For each value of 4, 8, 16, or 32, a set of system resources will be locked. Your database should have a page size of at least 32 KB. HPU will be faster if the pagesize parameter value is 4: 8: 16. This parameter is ignored with FixPak 3.

bufsize

This is the value (8388608) of the parameter defining the default buffer size that is used when writing the output file.The value is the actual number of bytes used for this buffer.

memsize

This parameter controls the default size of the memory buffer used between the reader and writer co-processes when reading DB2 files. This value is the actual number of bytes. If the input I/O is executed on the same device of the output (typical Windows demo situation) we recommend a pretty small memsize (10-20 KB) and the largest bufsize possible (say 60 MB) without causing paging. For a regular production system with more than one disk, a buff size of 5-10 MB, and a memsize of 2-4 table spaces extents should be sufficient. These recommendations are subject to change with the improvements that will be provided in the

future to the I/O subsystem of HPU, with the intent of aiming at the best performance on large configurations.

db2dbdft
This parameter corresponds to the database name used by HPU when no database name is given (command line option –d).

db2instance
The value that corresponds to the parameter is the instance name used by HPU when no run-time value is given (command line option –i).

maxunloads
This parameter limits the number of concurrent threads that HPU will generate when unloading DB2 files.HPU generates one thread by unload block.

maxsockets
This parameter is the number of connections the deamon will answer to concurrently. This value is not user tunable and does not set a limit on the number of sockets that HPU execution will open since they are HPU defined.

insthomes
This parameter provides the DB2 home directory path for all DB2 instances. At install time, only one DB2 instance is granted with HPU, and if you need to grant more instances, you must modify this parameter. For example, if you need to grant HPU with one default instance of db2inst1 (instance home is /home/db2inst1) and another instance of db2inst2 (instance home is /home/db2inst2), this parameter is:

```
insthomes=db2inst1:/home/db2inst1,db2inst2:/home/db2inst2
```

8.3 Using HPU for MP

This section describes the use of the HPU tool for distributed platforms. We discuss how to prepare your database before using HPU, the restrictions on the tool, and some examples on its use.

Preparing for unload

- The authority needed to perform the Load is any of the following:
  - sysadm
  - dbadm
- If unloading from the database object files, or if the use running the tool does not have the sysadm authority, ensure that DB2 database manager is running.

Restrictions

There are some restrictions on the complexity of SQL statements HPU can process. Currently, when these are encountered, HPU uses the DB2 Export utility and the options are not processed. Other restrictions are:

- HPU cannot perform an unload from a DMS table space backup.
- HPU does not support table joins.
- HPU cannot evaluate select statements involving a WHERE clause.
- HPU cannot select a subset of table columns.

8.3.1 Invoking HPU for MP

There are two ways of using the HPU tool:

- Command prompt
- HPU notebook in the Control Center

Note that the HPU is an executable only, and it cannot be used through an application programming interface.

Using the Command window

Use the following rules when you run HPU from your operating system command line using the **db2hpu** command and its options:

- Either the **-f**, **-t**, or **-v** option is required. All other command-line options are optional.
- Specify the full or the abbreviated option name.
  for example, either: **--database** or **--d** represents the database name.
- The option names must be lowercase whether you specify the full or abbreviated name.
- Unless the values enter for table_name, database_name, and password are enclosed in quotation marks, HPU converts them to uppercase in accordance with DB2 naming conventions.
- For options with associated values, you must separate the option names from the values with a space.

Simple unloads can be executed by specifying a few command line options:

```
db2hpu -d sample -t staff -o staff.del
```

In this example:

- data is unloaded from the staff table in the sample database,
- output is redirected to file staff.del,
- default SQL statement selects all data in the table,
- default data format is DEL, with column delimiter ‘,’ and char delimiter “”,
- data is unloaded from the database object files.

Settings needed to perform more complex unloads are defined through the unload configuration file. The control file option is specified by ‘-f’. Other command line options can be used in conjunction with it. To perform an unload reading the input values from file db2hpu.config issue:

```
db2hpu -f db2hpu.config
```

To unload a table from a backup create the following control file:

```
global connect to sample;
unload tablespace userspace1
backup "backup\sample.0\db2\node0000\catn0000\20021025\162850.001"
select * from mmilek.employee;
output ("employee.del" append)
format del;
```

In this example:

- All data from employee table is unloaded.
- Table resides in table space userspace1 in database sample.
- Path to the full backup image has to be provided.
- The output file, employee.del, is written in the append mode, default mode is replace.
- Output format is delimited ASCII.

To unload the whole table space:

```
global connect to sample;
unload tablespace sample.userspace1
output("userspace1.ixf")
lob in ("lobs")
lobfile ("userspace1.lob")
format ixf;
```

In this example:

- The whole table space userspace1 is unloaded using a single command.
- Single output file is created for each table
- userspace1.ixf is used as a basename, output files are of the form basename_schema_tablename
- LOB data is stored in external files, saved in the lobs directory
- Basename for LOB file names is userspace1.lob
- Output format is ixf

HPU tool creates a single file for each LOB object it unloads. If a large number of records containing LOB data needs to be exported both ‘lobs to’ and ‘lobfile’ can be specified. The tool can unload up to 1000 lob objects for each basename specified by the lobfile parameter. Multiple lob paths can be used to circumvent file system space limitations:

```
global connect to sample;
unload tablespace userspace1
select * from emp_resume;
output("emp_resume.ixf")
lob in ("lobs")
lobfile ("e_r1.lob", "e_r2.lob", "e_r3.lob", "e_r4.lob", "e_r5.lob", "e_r6.lob",
"e_r7.lob", "e_r8.lob", "e_r9.lob", "e_r10.lob")
format ixf;
```

In this example a maximum of 10000 records can be exported because 10 basenames for LOB files are specified,

HPU will call the Export API if the select statement is too complex for its processing:

```
global connect to sample;
unload tablespace userspace1
select deptnumb,deptname from org where DEPTNUMB<40;
output("org.del")
format del;
```

In this example:

- A subset of columns is unloaded to a del file (due to the WHERE clause)
- As HPU cannot process this select statement, the Export utility is used

HPU tool can be used to create a sampling of data contents:

```
global connect to sample;
unload tablespace userspace1
select * from employee;
maxrows 100 skip 5 interval 10
output("employee.del")
format del;
```

In this example:

- Every tenth record in the employee table is unloaded into the output file.
- First 5 records are skipped.

- A maximum of 100 rows is unloaded.

More examples of HPU use are distributed with the product. Please refer to Appendix A of the *IBM DB2 High Performance Unload for Multiplatforms Version 2, Release 1*, SC27-1623-01.

Using the Fast Unload Notebook of the Control Center

The screen shots in this section show the Control Center plugin distributed with HPU for MP V2.1, used with DB2 UDB V7. If you run a different version of HPU or DB2 UDB, graphical interfaces may not be exactly as shown:

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the **Tables** folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane.)
3. Right click the table you want in the contents pane, and select **Fast Unload** from the pop-up menu. You will be prompted for a username and a password. The Fast Unload Notebook opens.
4. Use the **Process Options** tab to set general unload parameters. These include HPU behavior if the select statement cannot be processed, partitions involved in unload, lock and quiesce options, number of concurrent unload tasks and the exit error code criterion. Figure 8-1 on page 205.
5. Use the **Output Options** tab to set column options, select sampling criteria and choose number and data-time formats. Figure 8-2 on page 205.
6. Use the **Large Objects** tab to set LOB options. Figure 8-3 on page 206.
7. Use the **Unload** tab to provide output and message file names, and to specify select blocks for the unload. Click the appropriate buttons to add (the Add Select Block window opens), change (the Change Select Block window opens), delete, and reorder multiple select blocks. Figure 8-4 on page 206.
8. Use the **Add Select Block** window or the **Change Select Block** window to set the output file name by clicking the **Add** button or the **Change** button. The Add Output File window or the Change Output File window opens and you can choose its format by clicking the **Options** button (the Format Options window opens). Figure 8-5 on page 207.
9. Use the Format Options window to set the output file type. If user format is selected, specify the column list by clicking the **Add** button (the Add Column Format window opens), the Change button (the Change Column Format window opens) or the **Remove** button. Figure 8-5 on page 207.
10. Use the Add Column Format window or the Change Column Format window to specify column type, delimiter, size, null indicators, and number format options for each output column. Figure 8-7 on page 208.
11. Contents of the control file created using current settings can be viewed by clicking on the **Show Command** button on the Fast Unload Notebook.

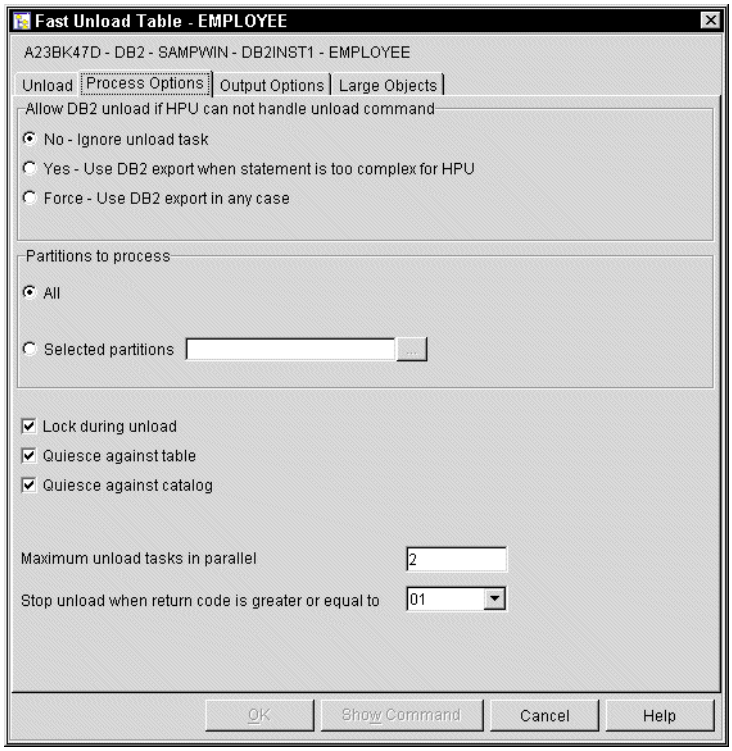

Figure 8-1 The Process Options tab of the Fast Unload Notebook

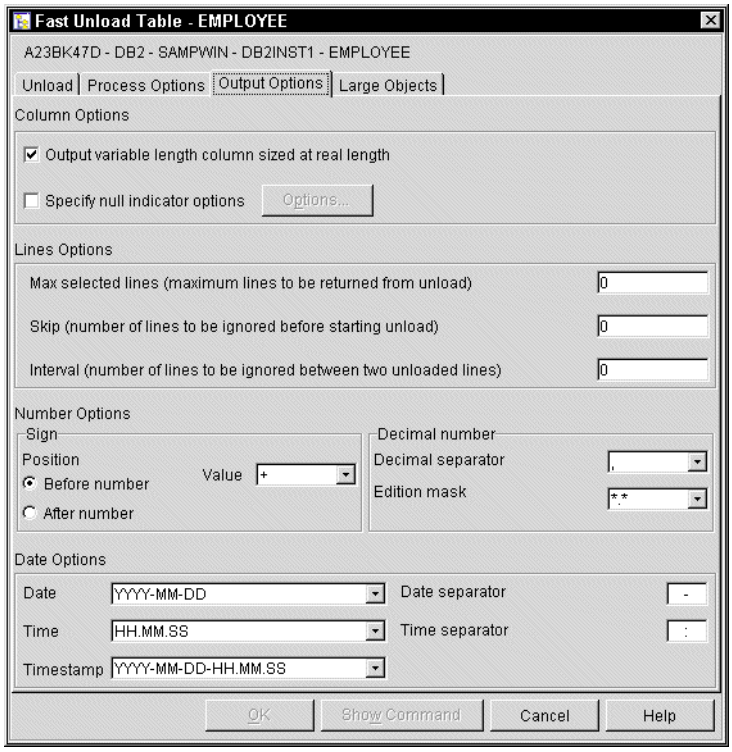

Figure 8-2 The Output Options tab of the Fast Unload Notebook

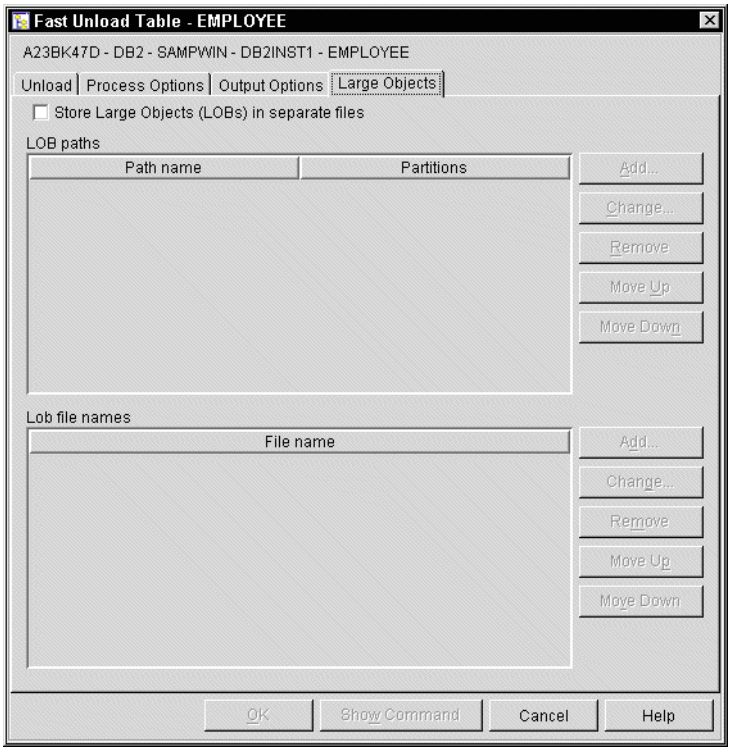

Figure 8-3 The Large Objects tab of the Fast Unload Notebook

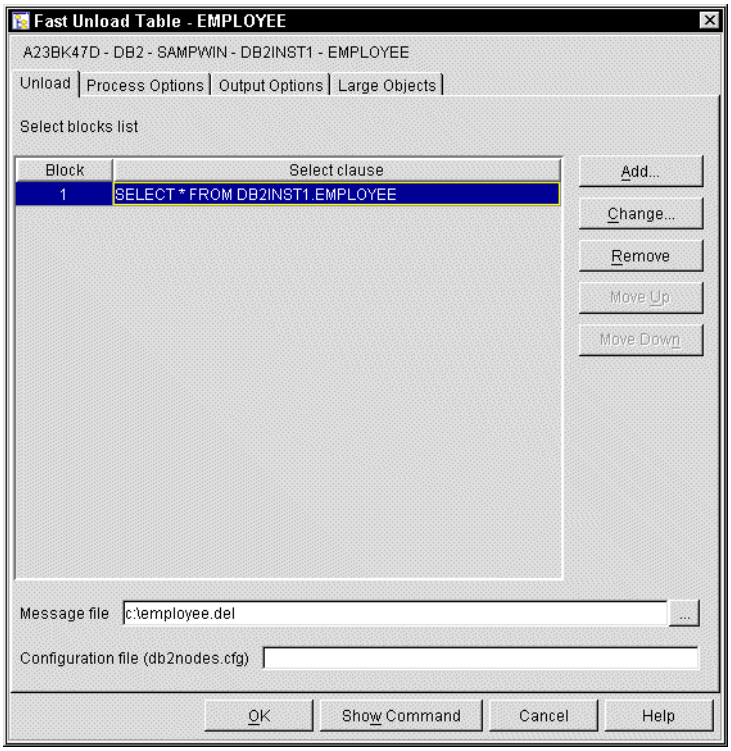

Figure 8-4 The Unload tab of the Fast Unload Notebook

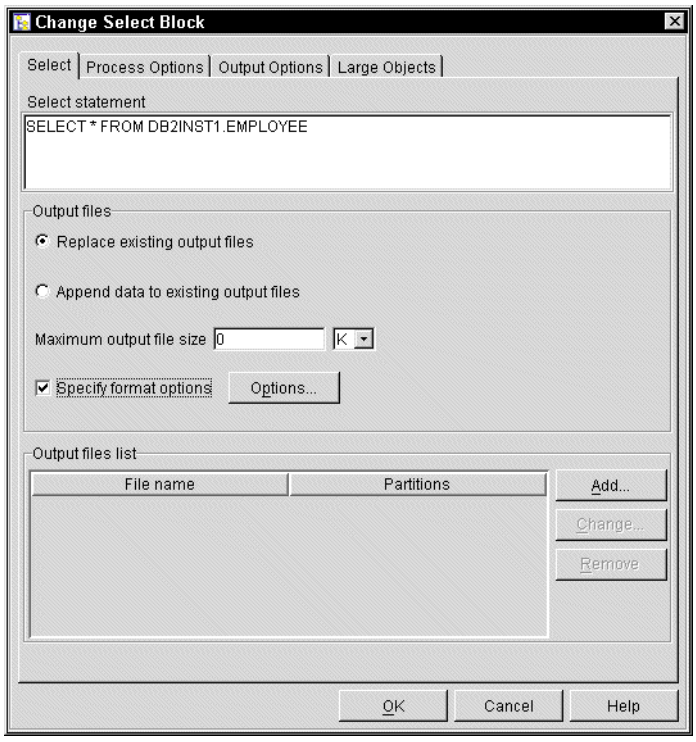

Figure 8-5 The Change Select Block window

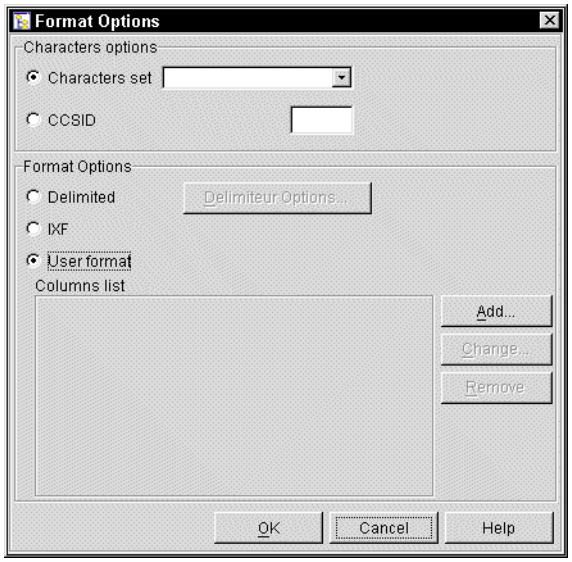

Figure 8-6 The Format Options window

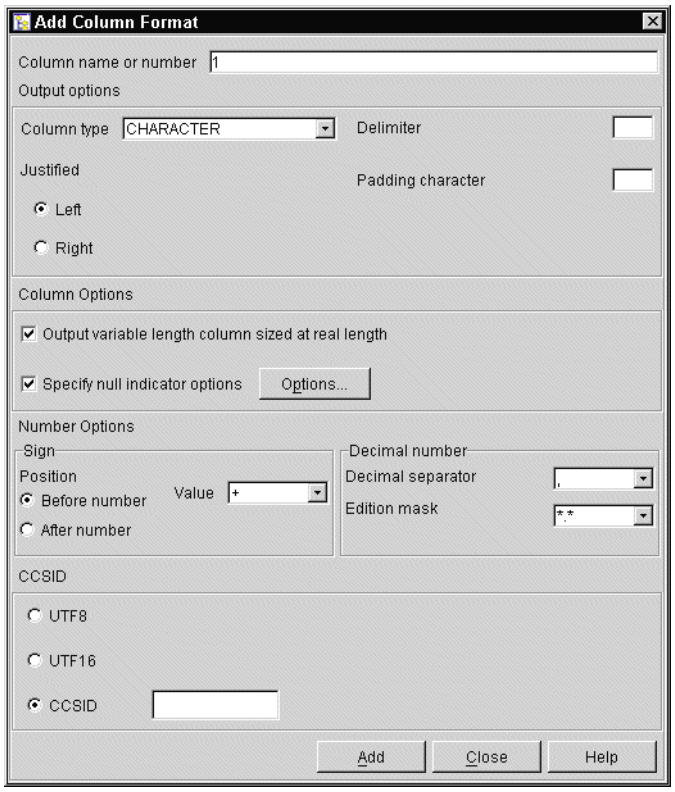

Figure 8-7 The Add Column Format window

If you need more information on the operation of the Control Center you can view the online Help facility inside the control center.

8.4 Comparing HPU for MP and Export

In this section we present performance comparisons between HPU for MP tool and Export utility.

DB2 sample database

In the first analysis we used tables from the sample database. Measurements were done on DB2 UDB V7 AIX and Windows platforms. The results are listed in Table 8-4, where the execution time is measured in seconds.

Tables from the sample database were populated with randomly generated data. LOB objects were stored in external files. Unless specified otherwise, Export was ran in the old lob mode, creating a LOB file for each LOB object.

By repeating some measurement we estimate a 5% error on the ratios of execution times in non-partitioned environments. Because of necessary network traffic, and a more complex processing model, this uncertainty is larger in partitioned environments. Our estimate here is 10%.

Table 8-4 Comparison of Export and Unload execution times on Windows platform

Table	Rows	Format	Export (s)	Unload (s)	Ratio
employee	1 M	IXF	29.0	17.7	1.6
employee	1 M	DEL	25.4	17.7	1.4
emp_resume	40 K	IXF	397.6	143.0	2.8
emp_resume	40 K	DEL	287.5	103.2	2.8

Non-partitioned sample database on Windows platform was used for the analysis summarized in Table 8-4 on page 209. Table employee does not contain any LOB or LONG columns and no indexes are defined on the table. Table emp_resume does contain a LOB column, and a two dimensional primary key is defined on the table.

HPU outperforms Export in all scenarios. The difference is more significant when LOB fields are present.

TPC-C like benchmark

In another set of measurements, a TPC-C like benchmark was used to compare HPU and DB2 Export. The results are reported in Figure 8-8.

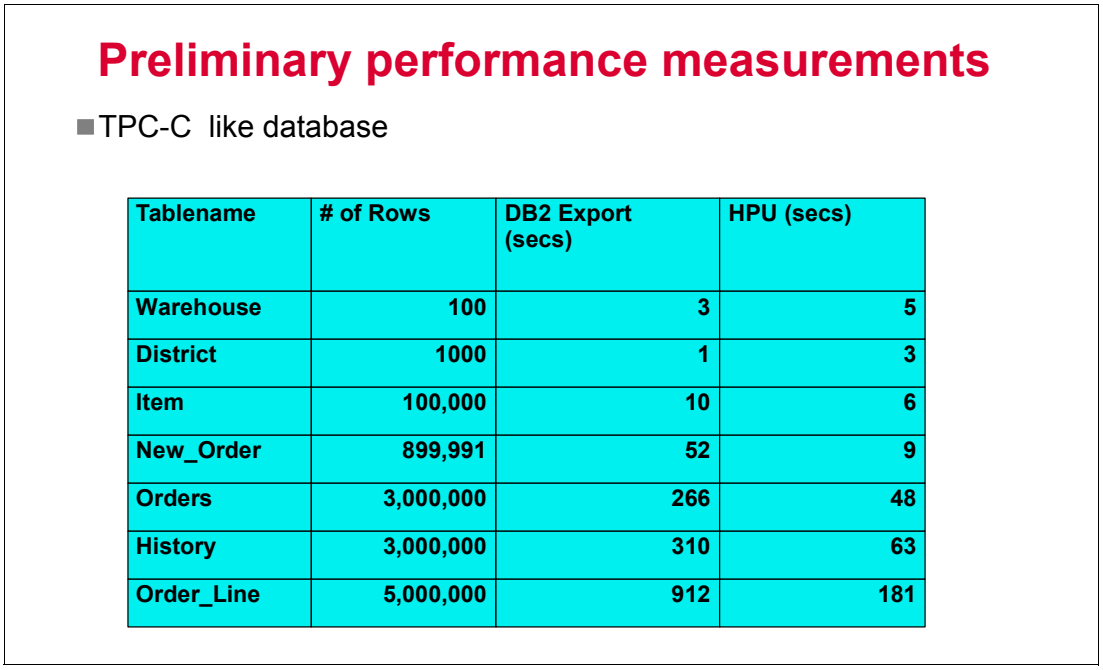

Figure 8-8 DB2 Export and HPU TPC-C measurements

The results have also been reported in the diagram in Figure 8-9.

These measurements show that HPU is better in elapsed time than DB2 Export. In this case the difference is small for a small number of rows, but it reaches a ratio of five times at around 1 million rows and remains the same up to five million rows. Note that the tables used in this set of measurements are not large and have rather narrow row length. They are significative for an OLTP like environment. More measurements are planned from other sets of applications.

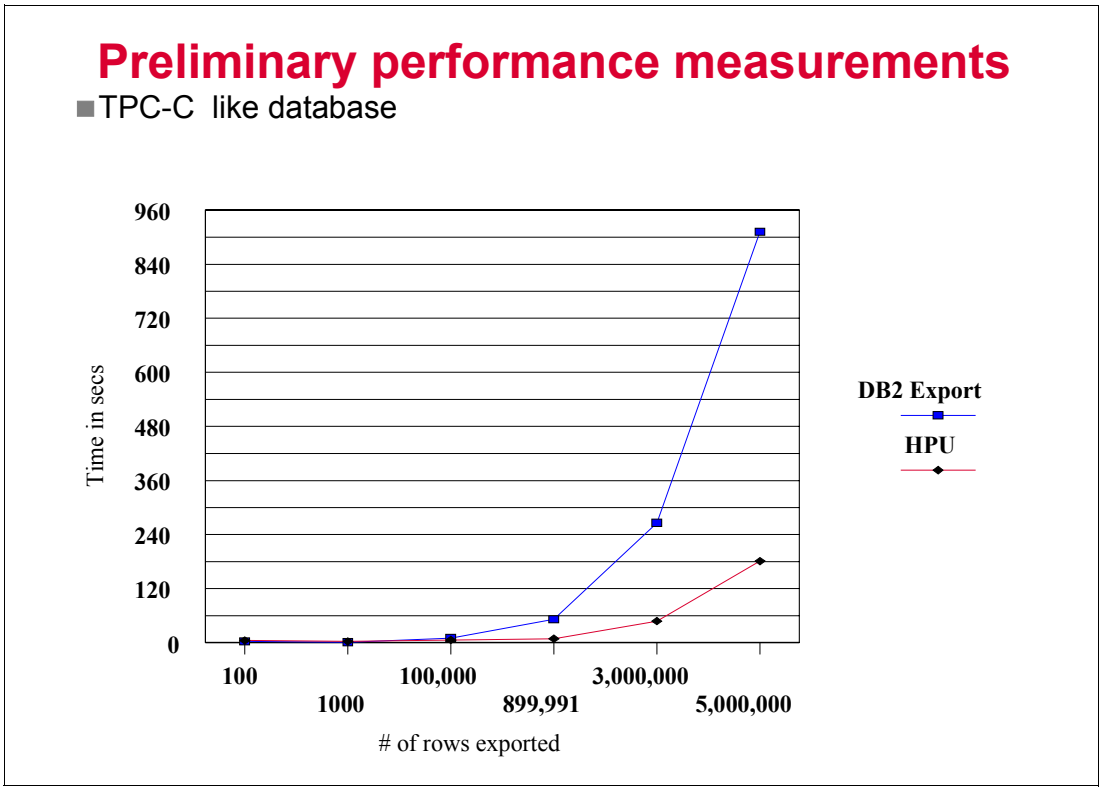

Figure 8-9 Chart of DB2 Export vs. HPU performance

8.4.1 When to use HPU for MP tool or the Export utility

This section summarizes strengths and weaknesses of the two methods used for moving data from the database into an external medium. These results should help your make an informed decision as to which utility/tool to use in a given scenario.

Use HPU when

- Performance is critical, and the select statement can be processed by HPU. As shown, HPU outperforms Export in most scenarios, sometimes by as much as a factor of 5.

  Before using HPU in production, use a subset of your data to verify that HPU does work properly in your setup.

Use Export when

- You need to use a select statement that HPU does not support.
- The select statement specified could use an index. HPU does not use all indices to access table data.
- You need to store LOB data in external files, and you run a version of DB2 UDB that supports exporting multiple LOB objects into a single LOB file.
- You need to unload a partitioned table into a single output file.
- You need to access data not located on the machine where HPU is running.

Part 4

Scenarios

In this part we describe some scenarios of moving data.

This part is structured in the following chapters:

- Getting ready for moving data
- Moving data to DB2 for z/OS
- Moving data to DB2 Distributed

9

Getting ready for moving data

This chapter describes the tools that can be used to extract the data definition language. We discuss these Data Management tools:

- Before moving data
- DB2 Administration Tool
- db2look command
- DB2 Control Center

9.1 Before moving data

Moving data can be a one time activity or it could be a continuous activity done periodically. There are several things you need to consider in moving data. You should know the proper tool to use, the software requirements, hardware requirements, authorities needed and the limitations you have on resources. In this section, we discuss some of these concerns. Our objective is present the information to you so that you can make an informed decision on your data moving.

In moving data across different databases, you also have to consider:

- Column definitions (primary keys, foreign keys, unique keys, etc.)
- Referential integrity
- Table index

9.1.1 Choosing a tool or utility

Database administrators weigh the pros and cons of each tool before using it. The advantages and disadvantages of each tool should be understood with respect to the environment and the circumstances in which it operates. The risks involved in moving the data, the performance objectives that need to be followed, and the resource constraints of the system set-up must be taken into consideration.

These are some factors you need to consider in choosing which tool is the best suited for your data migration objectives.

Speed

Performance is one of the most important factors you need to consider when choosing a tool. You need to know the length of time you have to complete the migration and the volume of data you need to move. There are a lot of benchmark testing done by different vendors, but you need to know that these tests are done in a controlled environment.

You need to understand the constraints and bottlenecks that you have in *your* system. Using the fastest tool or most efficient tool is useless if your system has a network or I/O bottleneck.

Some tools are simply built for speed. But nothing is for free. There are always trade-offs along the way. Versatility and control could be sacrificed to add a few mph on a tool. There are tools now that no longer use DB2 to unload. These tools may provide better performance because they directly access the physical file. However, they may lack the control and versatility that comes from executing the SQL statements.

Concurrency

This is the ability of a tool to move data in a live environment. This means you do not have to stop the database to load or unload the data. Furthermore, some tools have online capabilities, giving other applications concurrent access to the data being processed.

This ability is essential for database shops where continuous availability of data is of primary importance. For example, a bank's ATM needs a 24x7 access to data of the depositors accounts. A migration or back-up of a database needs to be concurrent to avoid down time.

Volume

The volume of data that can be transferred by a tool at a given time is an important factor that should be considered. This factor matters more if you are moving an entire database or table space involving terabytes of actual data.

Network capacity

The Network speed and capacity in transferring data needs to be considered in choosing a tool. For example, if the network that you will use in transferring data from one database to another is already congested, you should consider using tools that can write and read its output to and from a removable media.

Recovery

Recovery is the ability to go back to a consistent state and retrieve data if a database operation fails. For example, if loading data to an active database fails, you need to be able to recover by doing a rollback.

Capability of moving metadata, DDL, and DCL

Moving data requires movement not only of the actual data inside the tables, but also the related descriptions, such as contained in data dictionaries, and definitions; such as table definitions, the DDL or data definition language, and authorities and privileges, the DCL or data control language. You need to know what your tool is capable of moving. This will determine the tasks that you have to do before and after moving the data.

SQL capability

A tool that can execute SQL has a great advantage in versatility. The SQL select statement enables the tools to process data selectively. You can choose which columns you want, join tables, and have a condition for the rows you want to unload.

This is very useful if you intend to create a test database by using a subset of your data.

Sort capability

This is the capability of a tool to sort the data while doing the load or unload operation. If a certain load or unload sequence is required and this capability is not present in the tool, you need perform the data sorting separately, before performing the load. Without this capability data is loaded in the same sequence as it appears in the input file.

Referential integrity

Referential integrity, in the RDBMS sense, is a restriction imposed by the DBMS in changes in a table, so as to ensure that every column can only refer to a column that exists. The foreign key of a table needs to access an existing primary key or unique key of another table. If this concept is applied in moving data it would simply mean, moving a table (table A) whose column refers to a column of another table (table B), requires that both of the tables (table A and table B) need to be included in the data movement.

You need to know whether the tool you are using respects the referential integrity of your data. The referential integrity of your database is implemented through the DBMS or through the application that uses the data. If it is not implemented through the DBMS then you have to manually monitor that all tables needed to preserve the referential integrity are being downloaded together.

Security

Security in a data movement tool is its ability to control the access of people who can move or copy data from a database or table. Security of a database also necessitates that even the back-up files cannot be restored unless the person has the proper authority.

Capability to handle different encoding schemes and code pages

A tool that can move data across different platforms requires that it can handle different encoding schemes and it can convert from encoding scheme to another. Mainframe stores its data in EBCDIC while distributed platforms (UNIX, Windows NT, and Linux) store its data in ASCII. Hence, one consideration that you have to put in mind in moving data across different platforms is the EBCDIC to ASCII conversion. For details see 9.1.5, “Encoding scheme and code pages” on page 218.

Capability to handle different file formats

Being able to handle positional, delimited, and IXF format is an important consideration in moving data to another platform. The tool used to unload data from the source need to be able to produce an output format that can be read by a tool in the target platform. For example, if you are moving to a mainframe database you need to use a tool that can read positional format. For details see 9.1.4, “File format considerations” on page 217.

Capability to add dimensions to the unloaded data

There are some tools that allow the addition of new dimensions or columns on the output (typically a timestamp). These additional dimensions are columns added on the target table. It is generated through computation or processing done on the existing columns of the source database.

Capability to selectively unload data

A tool that can perform logical operations when unloading data is important when additional logical processing is needed to selectively unload data to be placed on the target database. This will enable you to save time in accessing the target database after the data is moved. This can be very useful in cleaning the data that you download. You can check whether a value in a column falls within the range of accepted values. Or, this can be useful if you just need to extract a subset of the source database and migrate it as a test database.

9.1.2 Disk space considerations

Make sure that you know how much disk space is required by the data that you are going to move. This space should be allocated before you start. The amount of disk space required will vary, depending on the complexity of the database (the number and size of database objects). These objects include all tables and views. You should make available at least two times the amount of disk space that the database currently occupies.

If your SYSCAT table space is an SMS type of table space, you should also consider updating the database configuration parameters that are associated with the log files. You should increase the values of *logfilsiz*, *logprimary*, and *logsecond* to prevent the space for these log files from running out (normally SQL1704N with reason code 3). If this happens, increase the log space parameters, and repeat the migration.

9.1.3 Software considerations

These are some things you have to consider in your software environment before you do database migration.

DB2 version compatibility

You need to consider if the DB2 version you are installing and migrating into can work properly with the old version that you have. The down-level compatibility becomes an issue only when you intend to use an older DB2 version with a new one.

9.1.4 File format considerations

File format compatibility is important when exporting, importing, or loading data across platforms. The following sections describe PC/IXF, delimited ASCII (DEL), positional ASCII (ASC) and WSF file format considerations when moving data between different operating systems.

PC/IXF file format

PC/IXF is the recommended file format for transferring data across platforms. PC/IXF files allow the Load utility or the Import utility to process (normally machine dependent) numeric data in a machine-independent fashion. For example, numeric data is stored and handled differently by Intel and other hardware architectures.

To provide compatibility of PC/IXF files among all products in the DB2 Family, the Export utility creates files with numeric data in Intel format, and the Import utility expects it in this format. However, take note that the PC/IXF format is different from the QMF IXF format that is used in the mainframe. These two formats are not compatible with each other. You can find documentation on their differences in the DB2 UDB Version 8 On-line documentation.

Depending on the hardware platform, DB2 products convert numeric values between Intel and non-Intel formats (using byte reversal) during both Export and Import operations.

PC/IXF files cannot be loaded, but it can be imported into a partitioned database environment using DB2 UDB V7.

Note: PC/IXF file format is different from the QMF/IXF format. These two file formats are not compatible with each other.

ASCII file formats

DB2 UDB tools and utilities support two different ASCII file formats:

- Delimited (DEL) for Load/Import/Export
- Positional (ASC) for Load/Import

DEL allows rows larger than 32 KB, while ASC does not.

ASCII files may have characteristics specific to the operating system on which they were created. These characteristics are:

- Row separator characters
  - UNIX based text files use a line feed (LF) character.
  - Non-UNIX based text files use a carriage return/line feed (CRLF) sequence.
- End-of-file character
  - UNIX based text files do not have an end-of-file character.
  - Non-UNIX based text files have an end-of-file character (X'1A').

When ASCII files are transferred from one operating system to another through FTP, changes to the data can occur.

Notes:

1. FTP can handle operating system-dependant differences if you transfer the files in text mode; the conversion of row separator and end-of-file characters is not performed in binary mode.
2. If character data fields contain row separator characters as actual data, these will also be converted during file transfer. This conversion may cause unexpected changes to the data. For this reason, it is recommended that you do not use ASCII files to move data across platforms. The PC/IXF format is recommended, since data is encoded in a binary format that is not subject to OS specific characters.

WSF file format

Numeric data in WSF format files is stored using Intel machine format. This format allows Lotus WSF files to be transferred and used in different Lotus operating environments (for example, in Intel based and UNIX based systems.)

As a result of this consistency in internal formats, exported WSF files from DB2 products can be used by Lotus 1-2-3 or Symphony running on a different platform. DB2 products can also import WSF files that were created on different platforms. Transfer WSF files between operating systems in binary (not text) mode.

Note: Do not use the WSF file format to transfer data between DB2 databases on different platforms, because a loss of data can occur. Use the PC/IXF file format instead.

For details see *DB2 UDB Data Movement Utility Guide and Reference Version 8*, SC09-4830.

9.1.5 Encoding scheme and code pages

An *encoding scheme* is a set of rules used to represent character data. All string data stored in a table must use the same encoding scheme, and all tables within a table space must use the same encoding scheme except for: global temporary tables, declared temporary tables, and workfile table spaces. Encoding schemes include:

Extended Binary Coded Decimal Interchange Code (EBCDIC)

EBCDIC is an encoding scheme used to represent character data.This is normally used by the mainframe platforms (z/OS, OS/390, VSE, OS/400). It is a group of coded character sets that consist of 8-bit coded characters. EBCDIC coded character sets use the first 64 code positions (X'00' to X'3F') for control codes and the range X'41' to X'FE' for graphic characters.

American Standard Coded for Information Interchange (ASCII)

ASCII is an encoding scheme used to represent characters. It is limited to 256 code positions. This is the file format normally used by distributed platforms (UNIX, Windows, Linux)

Unicode

A universal encoding scheme for written characters and text that enables the exchange of data internationally. It provides a character set standard that can be used all over the world. It uses a 16-bit encoding form that provides code points for more than 65,000 characters and an extension called UTF-16 that allows for encoding as many as a million more characters. It provides the ability to encode all characters used for the written languages of the world and treats alphabetic characters, ideographic characters, and symbols equivalently, because it specifies a numeric value and a name for each of its characters. It includes: punctuation marks, mathematical symbols, technical symbols, geometric shapes, and dingbats. Three encoding forms include the following:

UTF-8: Unicode Transformation Format, a 8-bit encoding form designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208.

UCS-2: Universal Character Set coded in 2 octets, which means that characters are represented in 16-bits per character.

UTF-16: Unicode Transformation Format, a 16-bit encoding form designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200.

With DB2 UDB Version 8, the Fixpack 2 code level will allow the creation of Unicode UTF-8 tables in any table space of a database of any other encoding.

Coded character set identifier (CCSID)

A CCSID is a two-byte, unsigned binary integer that uniquely identifies an *encoding scheme* and one or more pairs of character sets and code pages.

A coded character set is a set of unambiguous rules that establishes a character set and the one-to-one relationships between the characters of the set and their coded representations. It is a character set in which each character is assigned a numeric code value.

Code page

A set of assignments of characters to code points. In EBCDIC, for example, 'A' is assigned code point X'C1' and 'B' is assigned code point X'C2'. In Unicode, 'A' is assigned *code point* "U+0041". Within a code page, each code point has only one specific meaning. A code point is a unique bit pattern that represents a character. It is a numerical index or position in an encoding table used for encoding characters.

9.1.6 Moving data with DB2 Connect

If you are working in a complex environment in which you need to move data between a host database system and a workstation, you can use DB2 Connect; the gateway for data transfer from the host to the workstation, as well as the reverse. The DB2 Export and Import utilities allow you to move data from a host or AS/400 and iSeries server database to a file on the DB2 Connect workstation, and the reverse.

Note:

1. The data to be exported or imported must comply with the size and data type restrictions that are applicable to both databases.
2. To improve Import performance, you can use compound SQL. Specify the compound file type modifier in the Import utility to group a specified number of SQL statements into a block. This may reduce network overhead and improve response time.

Restrictions

With DB2 Connect, Export and Import operations must meet the following conditions:

- The only file type supported by DB2 Connect is PC/IXF.
- A target table with attributes that are compatible with the data must be created on the target server before you can Import to it. The **db2look** utility can be used to get the attributes of the source table. We discuss about generating the DDL for data transfer on Chapter 9, “Getting ready for moving data” on page 213. Import through DB2 Connect cannot create a table, because INSERT is the only supported mode.

- A commit count interval must not be specified for the Import operation. If any of these conditions is not met, the operation fails, and an error message is returned.

  Note: Index definitions are not stored on Export or used on Import. If you Export or Import mixed data (columns containing both single-byte and double-byte data), consider the following:

- On systems that store data in EBCDIC (MVS, OS/390, OS/400, VM, and VSE), shift-out and shift-in characters mark the start and the end of double-byte data. When you define column lengths for your database tables, be sure to allow enough room for these characters. Variable-length character columns are recommended, unless the column data has a consistent pattern.

Note: If you have DB2 UDB Enterprise Edition or Extended Enterprise Edition, DB2 Connect is already included in the package. It is installed automatically in your system when you install the DB2 UDB.

9.2 Extracting the data definition language

The DDL needs to be extracted from the source database before you can recreate the definitions on your target database. These are some tools that you can use to extract the DDL:

- DB2 Administration Tool for z/OS
- db2look command
  - Invoked through CLP
  - Invoked through DB2 Control Center

9.2.1 Using DB2 Administration Tool for z/OS to extract DDL

You can generate the data definition language of a database, table, table space, or view residing in z/OS through the DB2 Administration Tool.

Getting the DDL through the DB2 Administration Tool

These are the steps to get the DDL through the DB2 Administration Tool for z/OS.

Step 1

Logon to the ISPF panel using an ID with at least a Select privilege on the table or database that you want to access. After logging on, go to the DB2 Administration Tool by entering **ADM** on the main ISPF panel. See Figure 9-1.

```
Master Application Menu - SC63
Opt => ADM                                                      Sc => HALF

                                                          USERID - PAOLOR3
Enter SESSION MANAGER Mode ===> NO    (YES or NO)         TIME   - 13:04

  TP  TOPS     - TSO Operator Presentation Sample
  TN  TPNS     - Teleprocessing Network Simulator
  AP  APPC     - APPC Adminstrative Application
  PW  PRINTWAY - IP PrintWay
  IN  INFOPRT  - Infoprint Server
  CV  CICSVR   - CICS VSAM Recovery
  MQ  MQS      - MQSeries
  TE  TERSE    - TERSE/MVS
  VE  VSAMEDIT - VSAM Editor
  FT  NV FTP   - NetView File Transfer Program
  IX  IXFP     - IBM Extended Facilities Product (for RVA)
  FM  FILEMAN  - File Manager
  S2  SDF II   - SDF II Functions
  ADM DB2ADM   - DB2 Administration Tool V4 and Object Compare V2
  AT  DB2AUT   - DB2 Automation Tool

 Use UP and DOWN PF Keys or commands to scroll MENU.
```

Figure 9-1 The ISPF main menu panel

Step 2

Go to the DB2 subsystem where your source tables or database is located. You can type **S** corresponding to the DB2 subsystem name, or you can type it on the DB2 system name prompt. See Figure 9-2.

```
 DB2 Admin ------------------- Active DB2 Systems ------------------ Row 1 of 3
Command ===>                                                 Scroll ===> PAGE


This is a list of the active DB2 systems on this MVS system.

Enter:
DB2 system name ===>

Or select the one you wish to use, or press END to exit.

Sel DB2 System Description                                              Group
--- ---------- ------------------------------------------------------------ -----
s   DB2G
    DB2H
    DB7Y
****************************** Bottom of data ********************************
```

Figure 9-2 The ISPF Active DB2 Systems panel

Step 3

Go to the DB2 system catalog panel by entering **1** on the Option prompt. See Figure 9-3.

```
DB2 Admin -------------- DB2 Administration Menu 4.1.0 ------------------ 13:41
Option ===> 1

   1 - DB2 system catalog                                DB2 System: DB2G
   2 - Execute SQL statements                            DB2 SQL ID: PAOLOR3
   3 - DB2 performance queries                           Userid    : PAOLOR3
   4 - Change current SQL ID                             DB2 Rel   : 710
   5 - Utility generation using LISTDEFs and TEMPLATEs
   P - Change DB2 Admin parameters
  DD - Distributed DB2 systems
   E - Explain
   Z - DB2 system administration
  SM - Space management functions
   W - Manage work statement lists
  CC - DB2 catalog copy version maintenance
                                                                  More:     +
Interface to other DB2 products and offerings:
   I  - DB2I DB2 Interactive
  OC  - DB2 Object Comparison Tool
```

Figure 9-3 The DB2 Administration menu — Option 1

Step 4

Choose the database object that you want to be the source of the DDL to be generated. If you want a DDL for the table or view, choose **T.** If you want it for the whole database choose **D**. In this example, we will generate the DDL of a group of tables, so we chose **D**. See Figure 9-4.

```
DB2 Admin --------------------- DB2G System Catalog --------------------- 13:53
Option ===> D

                                                                    More:     +
Options:                                                  DB2 System: DB2G
   V - Volumes                                            DB2 SQL ID: PAOLOR3
   G - Storage groups              GA - Authorizations to storage groups
   D - Databases                   DA - Authorizations to databases
   S - Table spaces                SA - Authorizations to tables spaces
   T - Tables, views, and aliases  TA - Authorizations to tables and views
   X - Indexes
   C - Columns                     CA - Authorizations to columns
   Y - Synonyms
   P - Plans                       PA - Authorizations to plans
   K - Packages                    KA - Authorizations to packages
   L - Collections                 LA - Authorizations to collections
   M - DBRMs                       RA - Authorizations to resources
Enter standard selection criteria (An SQL LIKE operator will be used):
Name       ===>                    Grantor  ===>
Owner      ===>                    Grantee  ===>
In D/L/H   ===>                    Switch Catalog Copy  ===> N   (N/S/C)
And/or other selection criteria (option xC shows you columns for option x)
Column     ===>                    Operator ===>       Value ===>
```

Figure 9-4 The DB2 system catalog panel — Option S

Step 5

Select the database by entering **GEN** on the Select column corresponding to it. You can sort the database list by entering **SORT NAME** on the Options prompt. See Figure 9-5.

```
DB2 Admin -------------------- DB2G Databases -------------- Row 53 to 65 of 87
Command ===>                                                  Scroll ===> PAGE

Valid line commands:
 T - Tables  S - Table spaces  X - Indexes  G - Storage group  ICS - IC status
 DIS - Display database  STA - Start database  STO - Stop database  A - Auth
 ? - Show all line commands
                         Storage  Buffer          Created      Index
Select Name     Owner    Group    Pool       DBID By       T E Buffer Pool
       *        *        *        *             * *        * * *
------ -------- -------- -------- -------- ------ -------- - - -----------
       DSG24P11 SG24P1M  SG246420 BP1         319 BARTR3     E BP2
       DSG24P12 SG24P1M  SG246420 BP1         327 BARTR3     E BP2
       DSG24P13 SG24P1M  SG246420 BP1         328 BARTR3     E BP2
       DSG24P21 SG24P2M  SG246420 BP1         333 BARTR3     E BP2
       DSG24P22 SG24P2M  SG246420 BP1         334 BARTR3     E BP2
       DSG24P23 SG24P2M  SG246420 BP1         335 BARTR3     E BP2
       DSG25DOP SG25DOP  SG256420 BP1         330 BARTR2     E BP2
       DSHADOW  SYS1     SYSDEFLT BP1         271 PAOLOR1    E BPO
GEN    DSNDB04  SYSIBM   SYSDEFLT BPO           4 SYSIBM       BPO
       DSNDB06  SYSIBM                         6 SYSIBM     E BPO
       DSNDB07  PAOLOR5  SYSDEFLT BP1           7 PAOLOR5  W   BP2
       DSNRLST  PAOLOR5  SYSDEFLT BP1         256 PAOLOR5    E BP2
```

Figure 9-5 The DB2 system catalog panel

Step 6

Check the options in this panel. Mark the database objects that you want to include in the DDL by entering **Y.** See Figure 9-6.

```
DB2 Admin -------------- DB2G Generate SQL from DB2 catalog ------------- 19:11
Option ===>

 Generate SQL statements for database DSNDB04           DB2 System: DB2G
                                                        DB2 SQL ID: PAOLOR3
                                                                 More:     +
 SQL statement types to be generated from the DB2 catalog:
   CREATE DATABASE. . . . . : Y        GRANT access ON DATABASE . : Y
   CREATE TABLESPACE. . . . : Y        GRANT access ON TABLESPACE : Y
   CREATE TABLE . . . . . . : Y        GRANT access ON TABLE. . . : N
   CREATE VIEW. . . . . . . : N        GRANT access ON VIEW . . . : N
   CREATE INDEX . . . . . . : N        ALTER TABLE ADD FOREIGN KEY: N
   CREATE SYNONYM . . . . . : N        LABEL ON . . . . . . . . . : N
   CREATE ALIAS . . . . . . : N        COMMENT ON . . . . . . . . : N
   CREATE TRIGGER . . . . . : Y        REBIND PLAN/PACKAGE. . . . : Y
   CREATE STORAGE GROUP . . : Y        GRANT use OF STORAGE GROUP : Y

 New names/values for generated SQL: (leave blank to use current values)
   Object owner . . . . . :          Run SQLID. . . . . . . . . :
   Alloc TS size as . . . . : DEFINED  (DEFINED, USED, or ALLOC)
   Database name. . . . . :
   Storage group for TS . . :          Storage group for IX . . . :
   Target DB2 version . . . :          (Current DB2 version: 710)
```

Figure 9-6 The DB2 Generate SQL panel

You can see the other options by pressing F8. You can leave most of the options in its default value. You also have a choice of running the DDL through batch or running it through TSO. You can specify this option in the Execution Mode prompt in Figure 9-9. You can chose BATCH mode or TSO mode. If you chose batch mode a JCL job will be generated. You can edit this job and submit it anytime you want. If you chose the TSO execution mode, the command will be run automatically and the DDL will be re-created immediately.

```
DB2 Admin -------------- DB2G Generate SQL from DB2 catalog ------------- 19:11
Option ===>

 Generate SQL statements for database DSNDB04            DB2 System: DB2G
                                                         DB2 SQL ID: PAOLOR3
                                                                   More:   -
   Database name. . . . . . :
   Storage group for TS . . :          Storage group for IX . . . :
   Target DB2 version . . . :          (Current DB2 version: 710)
   Use Masking. . . . . . . : N        (Yes or No)

 Output file and execution mode:
   Add to work stmt list. . : N        (Yes or No)
   Data set name. . . . . . :
     Data set disposition . : OLD      (OLD, SHR, or MOD)
   Execution mode . . . . . : TSO      (BATCH or TSO)
   Commit statements per. . :          (Db, tS, Tb, All, None. Default is All)
   DB2 defaults handling. . :          (Keep, or Remove. Default is Keep)

 DB2 Command output file:
   Data set name. . . . . . :
     Data set disposition . : OLD      (OLD, SHR, or MOD)
```

Figure 9-7 The DB2 Generate SQL options panel

Step 7

if you chose the BATCH execution mode, a JCL job will be generated as shown on the panel (See Figure 9-8). You should review this job and submit. After you have submitted the job the DDL will be re-created if there are no errors in the JCL.

```
DB2 Admin -------------- Edit Generated JCL -------------- Columns 00001 00072
Command ===> SUB                                            Scroll ===> PAGE
Verify and submit JCL
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001 //PAOLOR3D JOB (999,PAOLOR3),'DB2 UTILITY',
000002 //*       RESTART=STEPNAME, <== FOR RESTART REMOVE * AND ENTER STEP NAME
000003 //         REGION=0M,NOTIFY=PAOLOR3,
000004 //         MSGCLASS=X,
000005 //         CLASS=A
000006 //*
000007 //*********************************************************************
000008 //*
000009 //* DB2 ADMIN GENERATED JOB
000010 //*
000011 //*********************************************************************
000012 //*
000013 //*************************************************************ADB2GEN***
000014 //* STEP GENSQL: GENERATE SQL FROM DB2 CATALOG
000015 //*********************************************************************
000016 //GENSQL   EXEC PGM=IKJEFT01,DYNAMNBR=100
000017 //STEPLIB  DD DISP=SHR,DSN=GOC.V2R1M0.SGOCLLIB
```

Figure 9-8 This JCL panel will be displayed if you chose the BATCH execution mode.

If you chose the TSO execution mode, the DDL will be created immediately. There will be some messages from the DDL generation that will be shown on your panel. Just press F3 until you reach the DDL statements as shown in Figure 9-9.

I

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       SYS02324.T193403.RA000.PAOLOR3.R0104953         Columns 00001 00072
Command ===>                                                 Scroll ===> PAGE
000018 -- ADB2GEN: Generate DDL for Database DSNDB04                        --
000019 --                                                                   --
000020 -------------------------------------------------------------------------
000021 --
000022 --
000023 -------------------------------------------------------------------------
000024 -- Database=DSNDB04    Stogroup=SYSDEFLT
000025 -------------------------------------------------------------------------
000026 --
000027   SET CURRENT SQLID='SYSIBM';
000028 --
000029   CREATE DATABASE DSNDB04
000030     BUFFERPOOL BP0
000031     INDEXBP    BP0
000032     STOGROUP SYSDEFLT ;
000033 --
000034   SET CURRENT SQLID='PAOLOR5';
000035 --
000036   GRANT CREATETAB,CREATETS
```

Figure 9-9 The DDL statement of the database object in the DSNDB04 database

9.2.2 Using db2look to extract DDL

The db2look tool extracts the required DDL statements to reproduce the database objects of a target database on a source database. This tool can also generate the required UPDATE statements used to replicate the statistics on the objects in a test database, as well as the update database configuration and update database manager configuration parameters; and the db2set statements so that the registry variables and configuration parameter settings on the test database match those of the production database. It is often advantageous to have a test system contain a subset of the production system's data. However, access plans selected for such a test system are not necessarily the same as those that would be selected for the production system. Both the catalog statistics and the configuration parameters for the test system must be updated to match those of the production system. Using this tool makes it possible to create a test database where access plans are similar to those that would be used on the production system.

You are required at least a select privilege on the table of database before you can use this tool. You do not need to connect to the database to execute, because this tool automatically makes the database connection if you are not yet connected.

Syntax of db2look

The parts of the command outside the curly brackets are required. Those inside the curly brackets are optional. The options are not positional. Hence, you can interchange the placement of the letters representing the options.

db2look -d *database name* {-u *creator* [-z schema name] -s -g -a -h -r -c
[-t *tablename*] -p -o -f *filename* -e -m -l -x [-i *userid*] [-w *password*] -f }

Description of the options

These are the options that you can use with the **db2look** command:

-d DBname

Alias name of the database that is to be queried. DBname can be the name of a DB2 UDB for UNIX, Windows, OS/2, or DB2 UDB for OS/390 database. If the DBname is a DB2 UDB for OS/390 database, the db2look utility will extract the DDL and UPDATE statistics statements for OS/390 objects. These DDL and UPDATE statistics statements are statements applicable to a DB2 UDB database and not to a DB2 for OS/390 database. This is useful for users who want to extract OS/390 objects and recreate them in a DB2 UDB database.

If DBname is an OS/390 database, then the db2look output is limited to the following:

- Generate DDL for tables, indexes and views
- Generate UPDATE statistics statements for Tables, columns, column distributions, and indexes

-u Creator

Limits output to objects with this Creator ID. If option **-a** is specified, this parameter is ignored. If neither **-u** nor **-a** is specified, the environment variable USER is used.

-z Schema name

Limits output to objects with this schema name.

-s
Generate a PostScript file.

Note:

- This option removes all LaTeX and .tmp PostScript files.
- Required non-IBM software: LaTeX, dvips.
- The psfig.tex file must be in the LaTeX input path.

-g
Use a graph to show fetch page pairs for indices.

Note:

- This option generates a filename.ps file, as well as the LaTeX file.
- Required non-IBM software: Gnuplot.
- The psfig.tex file must be in the LaTeX input path.

-a
When this option is specified the output is not limited to the objects created under a particular creator ID. All objects created by all users are considered. For example, if this option is specified with the **-e** option, DDL statements are extracted for all objects in the database. If this option is specified with the **-m** option, UPDATE statistics statements are extracted for all user created tables and indexes in the database.

Note: If neither **-u** nor **-a** is specified, the environment variable USER is used. On UNIX based systems, this variable does not have to be explicitly set. On Windows NT, however, there is no default value for the USER environment variable: on this platform, a user variable in the SYSTEM variables must be set, or a set USER=<username>must be issued for the session.

-h
Display help information. When this option is specified, all other options are ignored, and only the help information is displayed.

-r
When this option is specified in conjunction with the -m option, db2look does not generate the RUNSTATS command. The default action is to generate the RUNSTATS command. The **-r** option is ignored if the -m option is not specified.

-c
When this option is specified in conjunction with the -m option, db2look does not generate COMMIT, CONNECT and CONNECT RESET statements. The default action is to generate these statements.

The **-c** option is ignored if the **-m** option is not specified.

-t Tablename
Table name. Limits the output to a particular table.

-p
Use plain text format.

-o Filename
If using LaTeX format, write the output to filename.tex. If using plain text format, write the output to filename.txt. If this option is not specified, output is written to standard output. You can specify the directory of the output file here. Example: **`-o c:/output.txt`**

-e
Extract DDL statements for database objects. This option can be used in conjunction with the **`-m`** option. DDL for the following database objects are extracted when using the **`-e option`**:

- Tables
- Views
- Automatic Summary Tables (AST)
- Aliases
- Indexes
- Triggers
- User defined Distinct Types
- Primary Key, RI, and CHECK constraints
- User Defined Structured Types
- User Defined Functions
- User defined Methods
- User defined Transforms
- Federated objects (wrappers, sewers, nicknames, user mappings) with DB2 V8

> **Note:** The DDL generated by db2look can be used to recreate user defined functions successfully. However, the user source code that a particular user defined function references (the EXTERNAL NAME clause, for example) must be available in order for the user defined function to be usable.

-m
Generate the required UPDATE statements to replicate the statistics on tables, columns, and indexes. The **`-p`**, **`-g`**, and **`-s`** options are ignored when the **`-m`** option is specified.

-l
If this option is specified, then the db2look utility will generate DDL for user defined table spaces, nodegroups, and buffer pools. DDL for the following database objects is extracted when using the **`-l`** option:

- User defined table spaces
- User defined nodegroups
- User defined buffer pools

-x
If this option is specified, the db2look utility will generate authorization DDL (GRANT statement, for example.)

-i
userid

Use this option when working with a remote database.

-w password
Used with the **-i** option, this parameter allows the user to run db2look against a database that resides on a remote system. The user ID and the password are used by db2look to logon to the remote system.

-f
Use this option to extract configuration parameters and registry variables.

Note: Only configuration parameters and registry variables that affect the DB2 query optimizer are extracted.

Calling the db2look through the command prompt

The **db2look** command can be called through the AIX or DOS command prompt. Here we report some examples of calling the **db2look** through the command prompt.

Example 1

Generate the DDL statements for objects created by all users in the database SAMPWIN. The db2look output is sent to file C:\DDLOUT.SQL as text file:

```
db2look -d sampwin -a -e -o c:\ddlout.sql
```

The output of the above command is shown in Example 9-1.

Example 9-1 Sample output of db2look command

```
-- This CLP file was created using db2look Version 7.1
-- Timestamp: 11/20/02 12:58:31 PM
-- Database Name: SAMPWIN
-- Database Manager Version: DB2/NT Version 7.2.0
-- Database Codepage: 1252


CONNECT TO SAMPWIN;


------------------------------------------------
-- DDL Statements for table "DB2INST1"."ORG"
------------------------------------------------

 CREATE TABLE "DB2INST1"."ORG"  (
        "DEPTNUMB" SMALLINT NOT NULL ,
        "DEPTNAME" VARCHAR(14) ,
        "MANAGER" SMALLINT ,
        "DIVISION" VARCHAR(10) ,
        "LOCATION" VARCHAR(13) )
       IN "USERSPACE1" ;

-- (there are other CREATE TABLE statements here truncated to shorten the example...)

COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

Example 2

Generate the DDL statements for objects created by user DB2ADMIN in database SAMPWIN. The db2look output is sent to file DDLOUT.SQL as text file:

```
db2look -d sampwin -u DB2ADMIN -e -o ddlout.sql
```

Note that the creator name is case sensitive.

In this case the output will be very similar to Example 9-1, only limited to a subset of values.

Example 3

Generate the UPDATE statements to replicate the statistics for the tables and indexes created by user DB2ADMIN1 in database SAMPWIN. The output is sent to file ddlout.sql:

```
db2look -d sampwin -u DB2ADMIN -m -o ddlout.sql
```

The output of this command is shown in Example 9-2.

Example 9-2 db2look with the mimic option

```
-- This CLP file was created using db2look Version 7.1
-- Timestamp: 11/21/02 09:26:05 AM
-- Database Name: SAMPWIN
-- Database Manager Version: DB2/NT Version 7.2.0
-- Database Codepage: 1252


CONNECT TO SAMPWIN;

------------------------------------------------

-- Mimic Tables, Columns, Indexes and Column Distribution

------------------------------------------------

-- Mimic table ORG

RUNSTATS ON TABLE "DB2INST1"."ORG" WITH DISTRIBUTION;

UPDATE SYSSTAT.INDEXES
SET NLEAF=-1,
    NLEVELS=-1,
    FIRSTKEYCARD=-1,
    FIRST2KEYCARD=-1,
    FIRST3KEYCARD=-1,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=-1,
    CLUSTERFACTOR=-1,
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=-1,
    DENSITY=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

UPDATE SYSSTAT.COLUMNS
SET COLCARD=-1,
    NUMNULLS=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

UPDATE SYSSTAT.TABLES
SET CARD=-1,
    NPAGES=-1,
    FPAGES=-1,
    OVERFLOW=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

UPDATE SYSSTAT.COLUMNS
SET COLCARD=-1,
    NUMNULLS=-1,
    --SUB_COUNT=-1,
```

```
    --SUB_DELIM_LENGTH=-1,
    AVGCOLLEN=-1
WHERE COLNAME = 'DEPTNUMB' AND TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';
-- (SOME TABLES WERE REMOVED HERE TO SHORTEN THE EXAMPLE )
COMMIT WORK;

COMMIT WORK;

-- Mimic functions

UPDATE SYSSTAT.FUNCTIONS
SET ios_per_invoc= -1.0,
    insts_per_invoc= -1.0,
    ios_per_argbyte= -1.0,
    insts_per_argbyte= -1.0,
    percent_argbytes= -1,
    initial_ios= -1.0,
    initial_insts= -1.0,
    cardinality= -1.0;

COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

Example 4

Generate both the DDL statements for the objects created by user DB2ADMIN and the UPDATE statements to replicate the statistics on the tables and indexes created by the same user. The db2look output is sent to file ddlout.sql:

```
db2look -d sampwin -u DB2ADMIN -e -m -o ddlout.sql
```

Example 9-3 db2look for DDL and mimic option

```
-- This CLP file was created using db2look Version 7.1
-- Timestamp: 11/20/02 03:22:43 PM
-- Database Name: SAMPWIN
-- Database Manager Version: DB2/NT Version 7.2.0
-- Database Codepage: 1252


CONNECT TO SAMPWIN;

------------------------------------------------
-- DDL Statements for table "DB2INST1"."ORG"
------------------------------------------------

 CREATE TABLE "DB2INST1"."ORG"  (
        "DEPTNUMB" SMALLINT NOT NULL ,
        "DEPTNAME" VARCHAR(14) ,
        "MANAGER" SMALLINT ,
        "DIVISION" VARCHAR(10) ,
        "LOCATION" VARCHAR(13) )
       IN "USERSPACE1" ;

------------------------------------------------
-- DDL Statements for table "DB2INST1"."STAFF"
------------------------------------------------
```

```
 CREATE TABLE "DB2INST1"."STAFF"  (
        "ID" SMALLINT NOT NULL ,
        "NAME" VARCHAR(9) ,
        "DEPT" SMALLINT ,
        "JOB" CHAR(5) ,
        "YEARS" SMALLINT ,
        "SALARY" DECIMAL(7,2) ,
        "COMM" DECIMAL(7,2) )
       IN "USERSPACE1" ;


-- (SOME CREATE TABLES WERE TRUNCATED HERE OT SHORTEN THE EXAMPLE...)

------------------------------------------

-- Mimic Tables, Columns, Indexes and Column Distribution

------------------------------------------

-- Mimic table ORG

RUNSTATS ON TABLE "DB2INST1"."ORG" WITH DISTRIBUTION;

UPDATE SYSSTAT.INDEXES
SET NLEAF=-1,
    NLEVELS=-1,
    FIRSTKEYCARD=-1,
    FIRST2KEYCARD=-1,
    FIRST3KEYCARD=-1,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=-1,
    CLUSTERFACTOR=-1,
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=-1,
    DENSITY=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

UPDATE SYSSTAT.COLUMNS
SET COLCARD=-1,
    NUMNULLS=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

UPDATE SYSSTAT.TABLES
SET CARD=-1,
    NPAGES=-1,
    FPAGES=-1,
    OVERFLOW=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2INST1';

--(SOME UPDATE STATEMENTS WERE REMOVED HERE TO SHORTEN THE EXAMPLE...)

COMMIT WORK;

COMMIT WORK;

-- Mimic functions

UPDATE SYSSTAT.FUNCTIONS
SET ios_per_invoc= -1.0,
```

```
    insts_per_invoc= -1.0,
    ios_per_argbyte= -1.0,
    insts_per_argbyte= -1.0,
    percent_argbytes= -1,
    initial_ios= -1.0,
    initial_insts= -1.0,
    cardinality= -1.0;

COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

Invoking db2look through the DB2 Control Center

You can generate the DDL of your database objects in a DB2 UDB on Windows and UNIX platforms using the DB2 Control Center. The procedure is relatively simple. You have to open the Control Center and locate the database or table that you want the source of the DDL. See Figure 9-10.

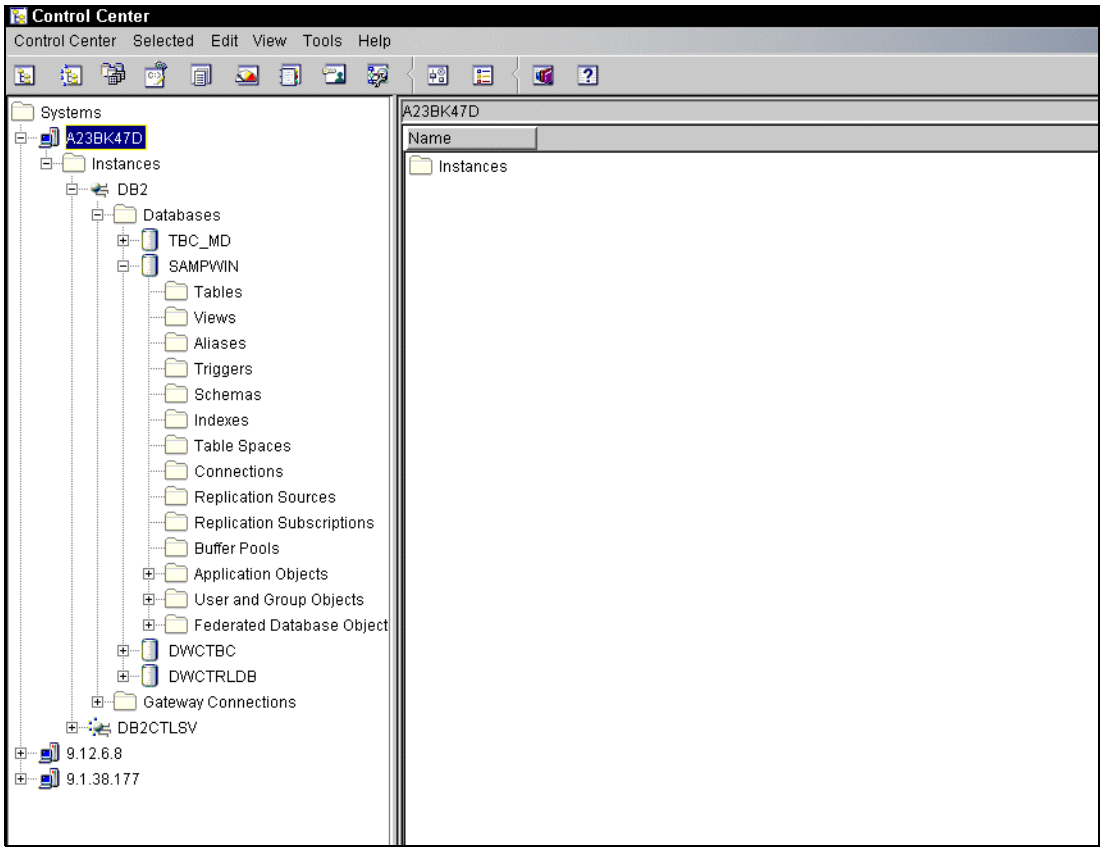

Figure 9-10 DB2 Control Center in Windows

Generating DDL for entire database

You also have the option to generate the DDL of the entire database. To do this, you need to right-click on the database name from the control panel window. See Figure 9-11.

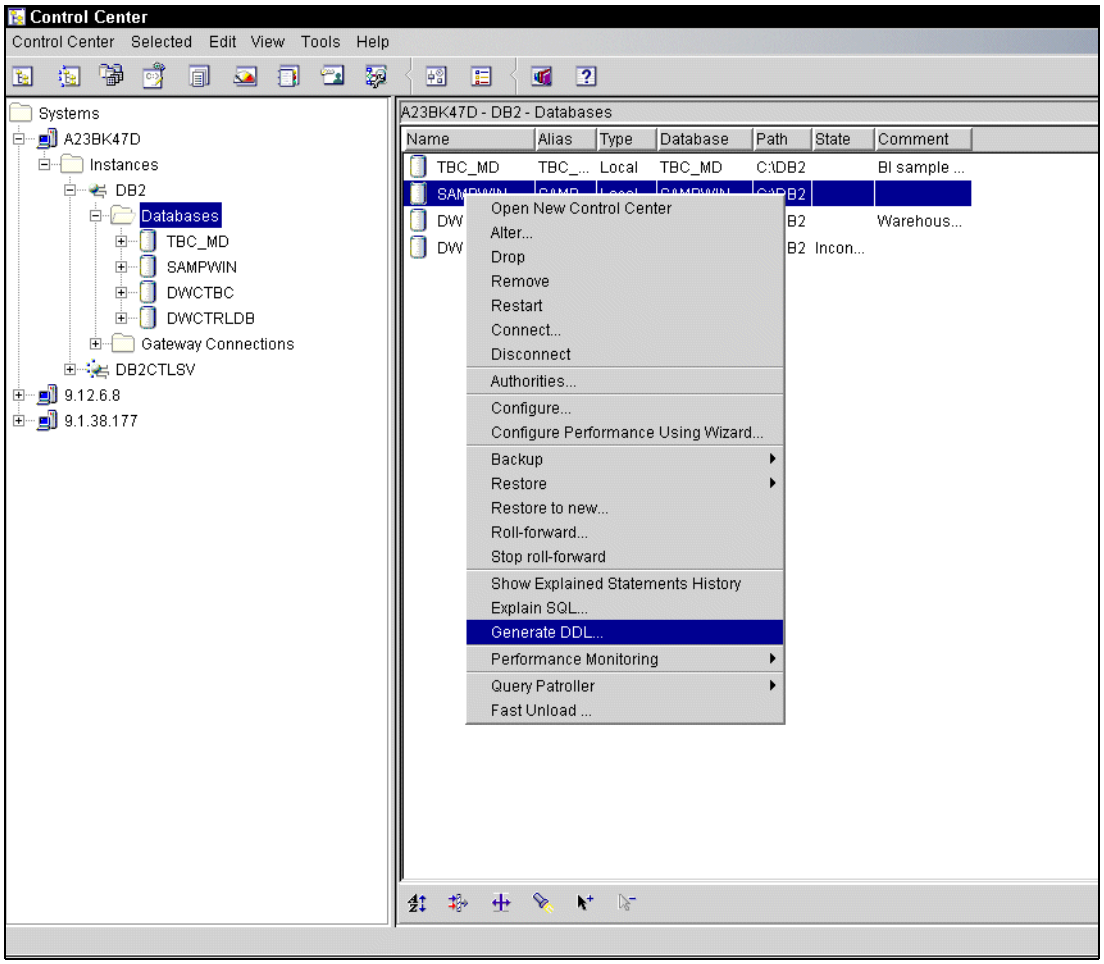

Figure 9-11 Option to generate DDL

Generating DDL for a single table

From the Control Center screen, right-click on the object that you want to perform the DDL extraction. You can chose the table name and right-click on it. A menu box will appear. Choose **GENERATE DDL...** from that menu box. See Figure 9-12.

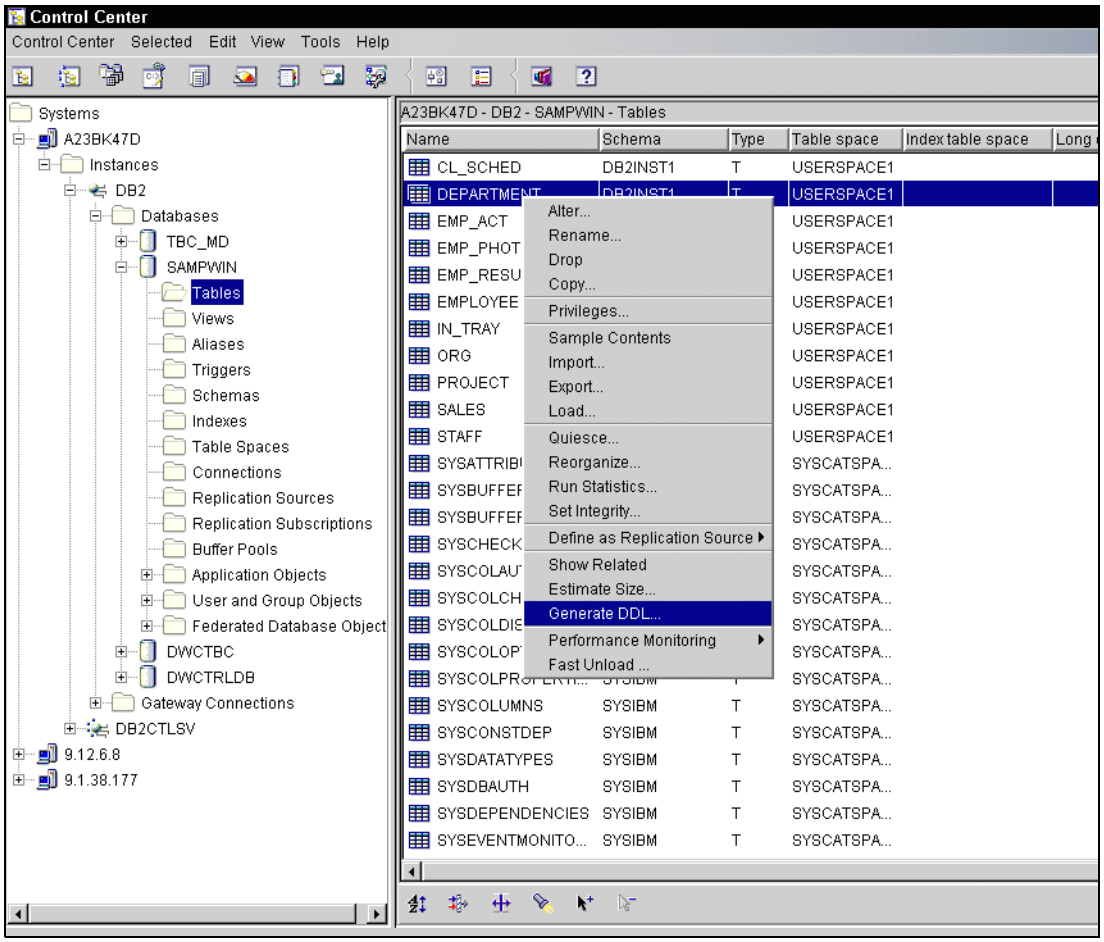

Figure 9-12 Generating DDL for a table

After choosing the **GENERATE DDL...** from the menu box, a window will pop-up. You can enter the database objects that you want to be included in your DDL by clicking the check-box. Then click the **Generate** button. See Figure 9-13.

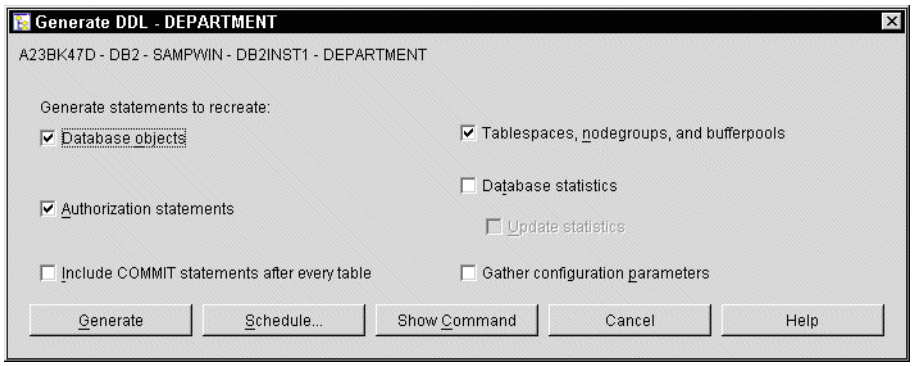

Figure 9-13 Generate DDL option box from the control panel.

You can look at the DB2 command that will be executed in the background by clicking the **Show Command** button. This is optional but you can see from this screen the db2look command being executed with the options that you selected using the check boxes. See Figure 9-14.

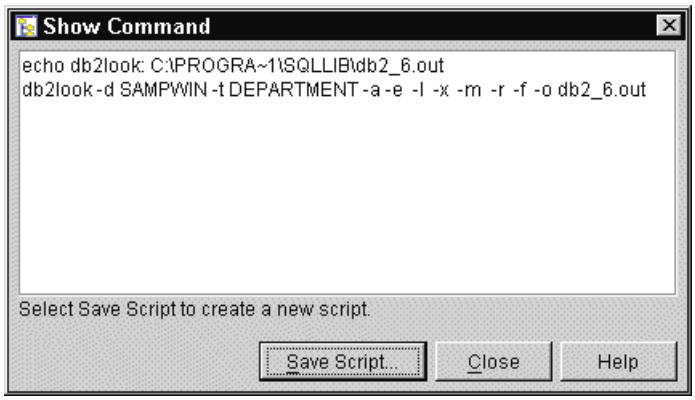

Figure 9-14 db2look command being executed in the background

After you click the **Generate** button on the options box. You will be prompted for your userid and password. Remember that you need at least a select privilege on the catalog tables.

After entering the userid and password, you will see a message box saying that the job is being run and you can see it in the Journal Window of the control center. Take note of the job number. See Figure 9-15.

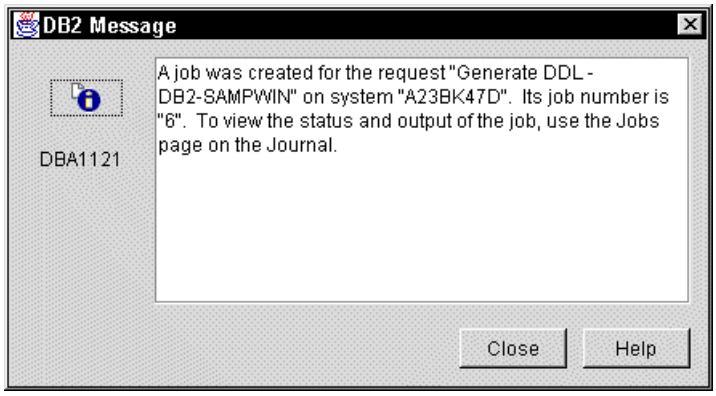

Figure 9-15 Message box stating the job number.

Go to the Journal window by clicking **Tools** in the menu bar and then **Journal** (**Tools** -> **Journal**). When you are in the Journal window click the **Job History** button. All the jobs will be shown in this window. Choose the job number that was specified in the message box. See Figure 9-16.

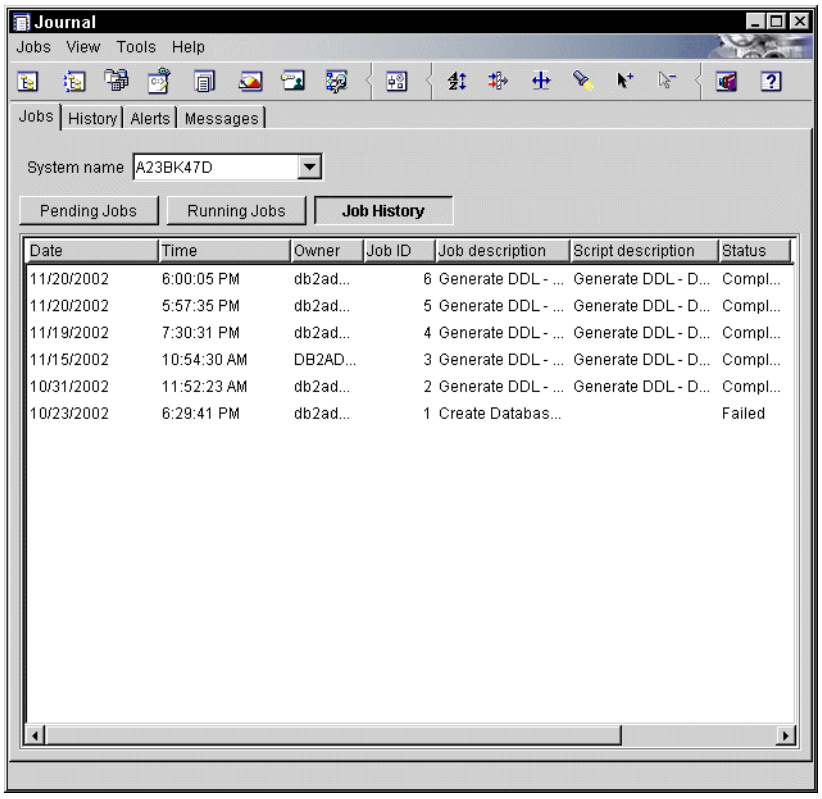

Figure 9-16 Journal window

After right-clicking on the line corresponding to the job number, a menu box will appear. Chose the **Show Results** option. See Figure 9-17.

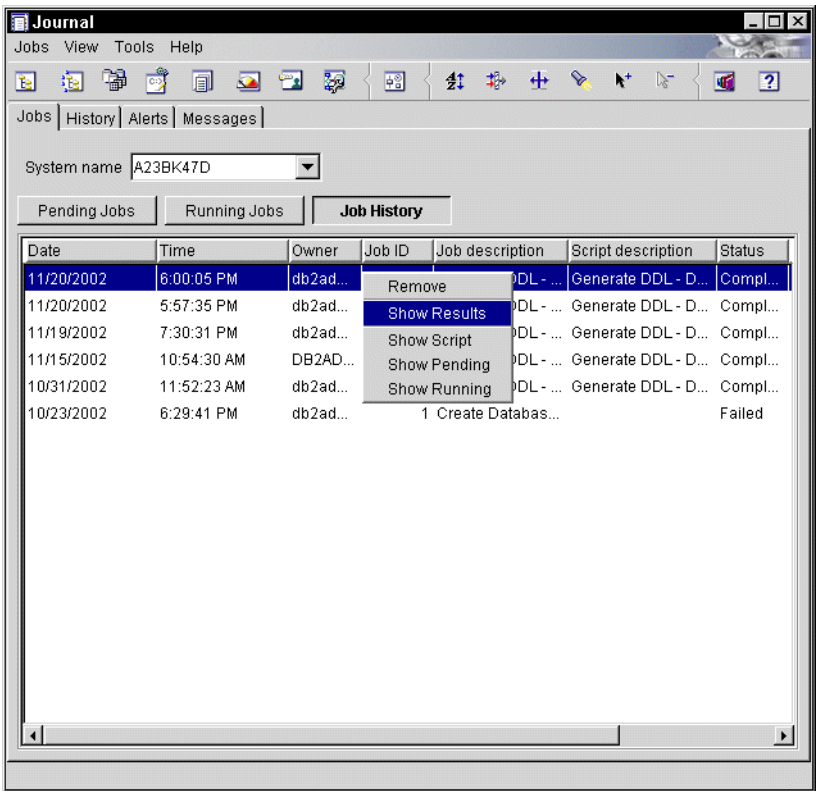

Figure 9-17 Chose the 'Show Results' option

The DDL will be shown in the job output screen. See Figure 9-18.

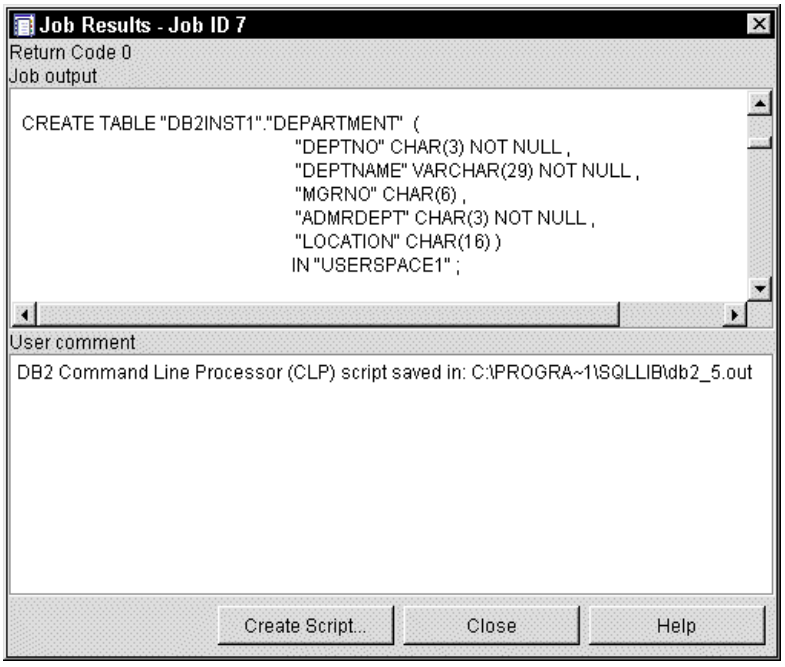

Figure 9-18 Generated DDL

9.2.3 Considerations

You can generate the DDL of a database objects through the **db2look** command, DB2 Administration Tool, or the DB2 Control Center. The DDL that you generated can be used as a skeleton to re-create the database objects of the source database on the target database. In the examples that we provided, we extracted the DDL from distributed and mainframe sources.

For DB2 Administration Tool

The DDL that was generated from the Admin Tool does not need to be modified if your target DB2 database also resides in the z/OS platform. You need to modify the DDL if you are transferring from a DB2 for z/OS database to a DB2 for distributed database. There is a minor difference in their DDL syntax.

For db2look

The **db2look** command can be called through the command prompt or through the DB2 Control Center. The DDL generated by this tool can be used without modification to create database objects on another DB2 distributed platform. However, as mentioned above, you need to do some minor modifications if you will use it to create DB2 database objects in a z/OS platform.

10

Moving data to DB2 for z/OS

In this chapter we discuss moving data to DB2 for z/OS. The source of data would be from either DB2 UDB on distributed platforms, or from DB2 for z/OS. We explore the different tools that we can use, and we give recommendations on which are the best tools to use and methods for doing the data transfer.

We use these tools for moving data to a DB2 for z/OS target:

- DB2 Cross Loader
- DB2 Data Propagator
- DB2 Unload and Load
- DB2 HP Unload and Load
- DB2 Export and Import
- SQL Insert with subselect (in DB2 Version 8 only)

There are other set of tools that we can use to move data to and from a mainframe database, such as DSNTIAUL and LOAD, but we are not going to write about them because there is already an extensive literature about the use these tools. If you need instructions on how to use them, you can refer to the *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*,SG24-6289-00, or the *DB2 UDB for OS/390 and z/OS V7 Utility Guide and Reference*, SC26-9945-03.

10.1 Overview of moving data to DB2 for z/OS

There are several DB2 utilities and tools that you can use to move data from a DB2 database in a distributed platform to DB2 database in z/OS. We are going to discus these tools and utilities, specifically on how they can be used as pairs in moving data. Presenting one tool as the thrower and the other as the catcher of data would provide a holistic view in describing the roles performed by these tools in moving data. We will give examples on the Cross Loader, Data Propagator, HPU and Load, Import and Export, SQL Insert with subselect. We believe these are the best DB2 tools and utilities to move data across the DB2 Family.

In moving data across distributed and mainframe platforms, DB2 Connect and the Distributed Data Facility (DDF) needs to be set-up in the system. See Appendix B, “DB2 connectivity” on page 307 for more information. The only exception to this rule is when you move data through FTP. Text files exported from a DB2 table in a distributed platform can be FTPed. These files can be converted as sequential files with EBCDIC format in a mainframe. You can then use these sequential files as input data to load tables in DB2 for z/OS. See Figure 10-1.

In DB2 Version 8 you can use Federated Database set-up in moving data across different platforms or even different relational databases. See Appendix A, “Defining a Federated Database” on page 299 for more information. In this scenarios, data can be moved from one distributed database to a host database, or vice-versa by simply using SQL Insert and subselect.

Important: When you move data, both the Data Definition Language (DDL) and the actual data needs to be moved to the target system. The examples here show how to move actual data. It is assumed that DDL has been transferred and tables have been created on the target database. See Chapter 9., “Getting ready for moving data” on page 213 for examples on extracting DDL and recreating it.

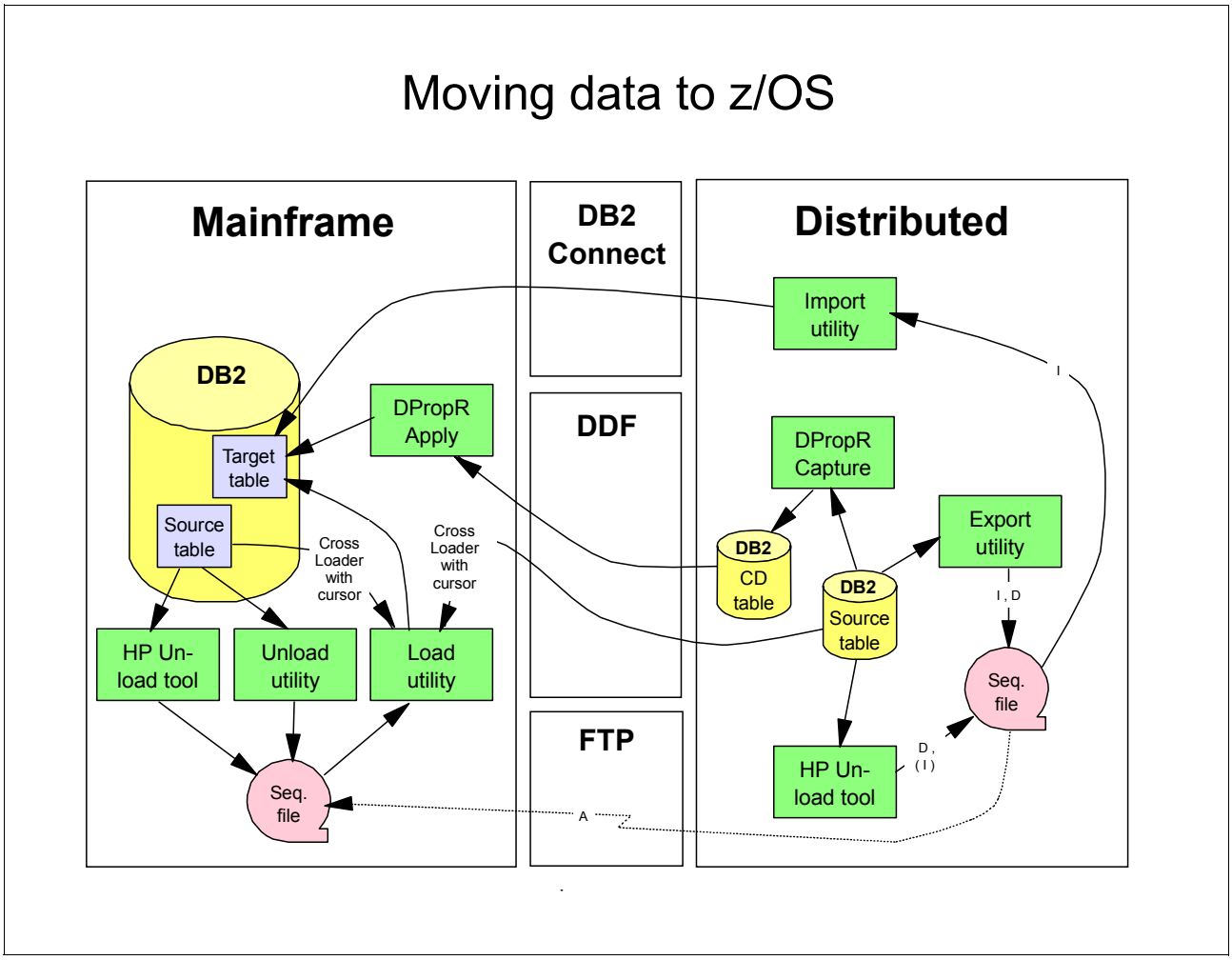

Figure 10-1 Diagram of moving data from DB2 on distributed to DB2 for z/OS

The letters written on the arrows represents the formats that you can choose between:

A	Positional (fixed) format (ASC)
D	Delimited format (DEL)
I	Integrated exchange format, PC version (IXF)
(I)	We were not able to produce IXF from HPU for MP

Note: The broken line means that for the time being there is no utility or tool which produces the positional format that LOAD on z/OS requires.

In this chapter, we use the Federated Database, DB2 Connect, and Distributed Data Facility (DDF) to move data from distributed platforms to mainframe. In Figure 10-2, you can see that we created a nickname for the PAOLOR7.DEPT and PAOLOR2.DEPT tables. Their nicknames in the Federated Database system are DB2ADMIN.DEPT390S and DB2ADMIN.DEPT390T, respectively. The DRDA WRAPPER enables the Federated Database, which is created in the DB2 distributed database, to *call* tables on a DB2 on z/OS database using nicknames. Take note that the table PAOLOR7.DEPT and the PAOLOR2.DEPT tables are not created in the Federated Database. Nicknames are created in the SAMPAIX8 database that point to tables in the DB2 for z/OS database.

We can also see in the diagram that Import and Export are executed on the DB2 distributed side, but it can write data on DB2 for z/OS. The HPU for z/OS, Unload and Load are all executed in the mainframe side, but they can get data from DB2 for distributed tables and write it on the DB2 for z/OS tables.

The Cross Loader is simply a Load from a cursor. The cursor can be created through a Select statement against a DB2 on distributed table. The result set will be loaded on a DB2 for z/OS table.

In moving data within DB2 for z/OS databases, we used Unload and HPU for z/OS to unload the data. We used Load utility on the mainframe to load the data to DB2 on z/OS tables. Obviously, we did not used the Federated Database system and DB2 Connect in this scenario.

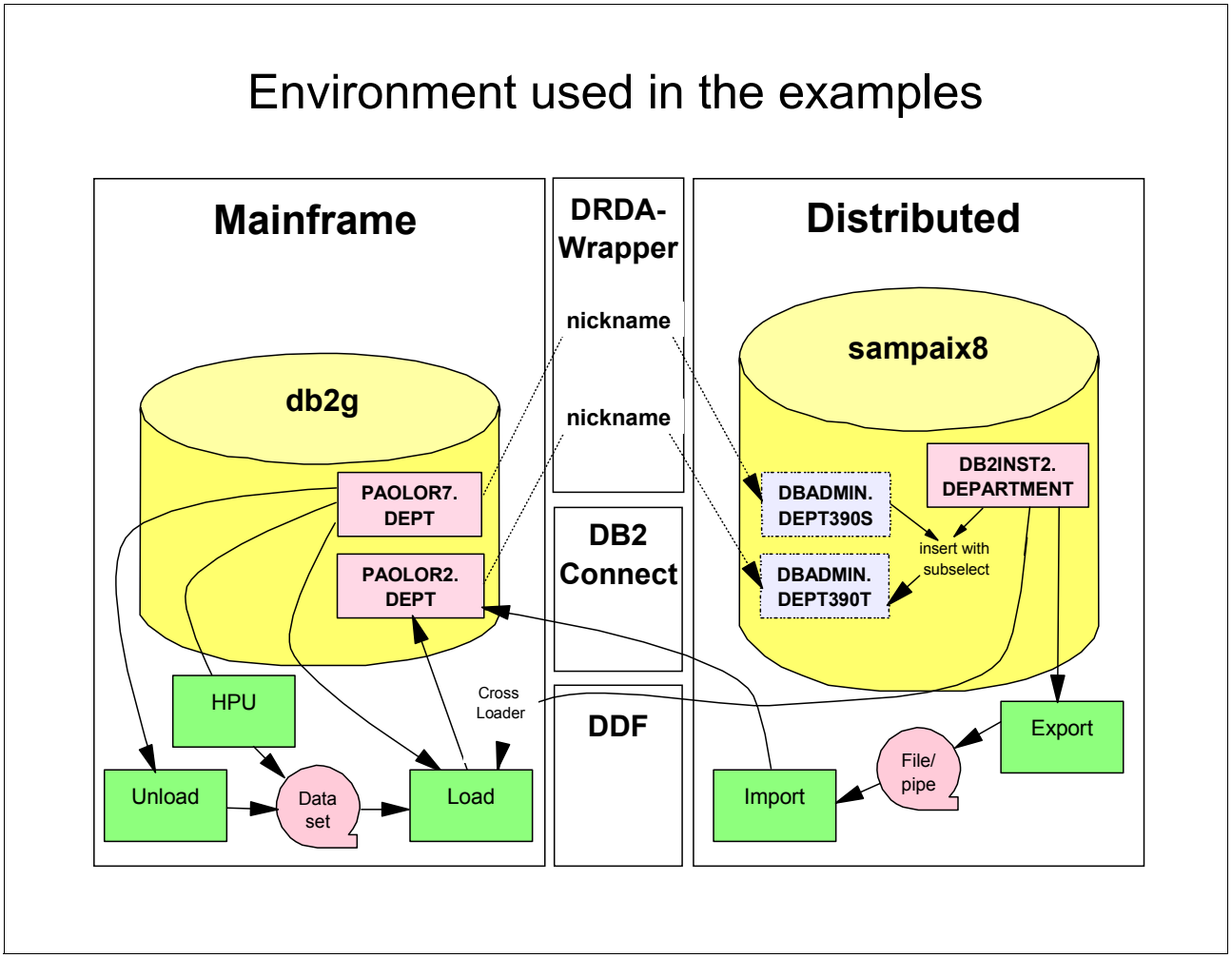

Figure 10-2 This is a diagram of the examples used in this chapter

The numbers in Table 10-1 represent the section where the function and the example are reported.

Table 10-1 Cross reference function to section in this chapter

Function	Section
Cross Loader	10.2.1, “Using Cross Loader to move data from DB2 distributed to DB2 for z/OS” on page 247 and 10.3.1, “Using Cross Loader to move data from/to DB2 for z/OS” on page 251
HPU	10.3.4, “HPU for z/OS and Load” on page 255
Unload	10.3.3, “Unload and Load” on page 252
Load	10.3.3, “Unload and Load” on page 252
Import	10.2.3, “Export and Import” on page 249
Export	10.2.3, “Export and Import” on page 249
Insert with subselect	10.2.4, “SQL Insert with subselect in a Federated Database” on page 250 and 10.3.5, “SQL Insert with subselect in a Federated Database” on page 264

10.2 Moving data from DB2 distributed

We are recommending these tools for moving data from DB2 on distributed platforms to DB2 for z/OS:

- Cross Loader
- Data Propagator
- Export and Import
- Insert with sub-select using Federated Database (only for DB2 UDB Version 8)

10.2.1 Using Cross Loader to move data from DB2 distributed to DB2 for z/OS

DB2 Cross Loader lets you transfer data from one database location to another in just one job. The data is read from the source location with a dynamic SQL statement and loaded in a table at the target location by the LOAD utility. The Cross Loader is actually an enhanced function of the LOAD utility. It is the ability of the LOAD utility to populate a table using data in a cursor you declared using EXEC SQL as input.

The advantage of this tool is its ease of use. You do not have to concern yourself with ASCII to EBCDIC conversions, or what file format you are going to use. Data is extracted from one table and loaded to another table in just one JCL step. There are no external files created in the process. Data is just moved internally from the source table to the target table.

You would need to define the remote database in the DDF and CDB of the mainframe. The remote database needs to be included in the SYSIBM.LOCATIONS, SYSIBM.IPNAMES, and SYSIBM.USERNAMES. You can only use the three part name when the remote database is defined in these tables.

To use DB2 Cross Loader, you need PTF UQ55541 for APAR PQ45268 and PTF UQ55542 for APAR PQ46759. These PTFs together with their PREREQ PTFs update DB2 V7 for OS/390 to have the Cross Loader functionality.

Moving data from DB2 distributed to z/OS using DB2 Cross Loader

In this example, we move data from a table in DB2 distributed database to DB2 on mainframe. We are using the three part name to access the table from the distributed platform (SAMPAIX.DB2INST2.DEPARTMENT). Here, SAMPAIX8 is the database name, DB2INST2

is the schema name and DEPARTMENT is source table name. The target table is DEPT and the schema name of the target table is PAOLOR2. See Example 10-1.

Database objects

The database object used in this example are in Table 10-2.

Table 10-2 Database objects for Example 10-1

Objects	Object name
Source DB2 on AIX	
Database name	SAMPAIX8
Schema name	DB2INST2
Table name	DEPARTMENT
Three part name	SAMPAIX8.DB2INST2.DEPARTMENT
Target DB2 for z/OS	
Subsystem name	DB2G
Schema name	PAOLOR2
Table name	DEPT

The Load option used in this example is explained in Table 10-3.

Table 10-3 Options used in Load

Option	Description
REPLACE	With the LOAD REPLACE option, first, the existing rows in table PAOLOR2.DEPT are deleted, and then all the rows read via cursor from table SAMPAIX8.DB2INST2.DEPARTMENT are newly loaded in PAOLOR2.DEPT. Other Option: RESUME YES, RESUME NO

DB2 Cross Loader sample job

This DB2 Cross Loader example, shown in Example 10-1, moves (copies) data from SAMPAIX.DB2INST2.DEPARTMENT in AIX to PAOLOR2.DEPT in z/OS.

Example 10-1 Cross Loader example of moving data from DB2 distributed to z/OS

```
//PAOLOR2A JOB (ACCOUNT),'PAOLOR2',NOTIFY=&SYSUID,USER=PAOLOR2
// EXEC PGM=DSNUTILB,PARM='DB2G,CRLOAD'
//STEPLIB DD DSN=DSN710.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
EXEC SQL
  DECLARE C1 CURSOR FOR
  SELECT *
  FROM SAMPAIX8.DB2INST2.DEPARTMENT
ENDEXEC
LOAD DATA
  REPLACE
  INCURSOR C1
  INTO TABLE PAOLOR2.DEPT
```

```
//*
```

10.2.2 Data Propagator

DB2 Data Propagator can also be used to move data from DB2 distributed to z/OS. Read 11.5, “Data Propagator” on page 279.

10.2.3 Export and Import

You can use the Import and Export utilities to move data from distributed DB2 databases to DB2 for z/OS databases. Both the Export and the Import utility are installed in the client side. There are no DB2 tools being used in the mainframe side. Before you can use this you should Bind the list of packages for the utilities in the DB2 client. The list of packages is found in the db2ubind.lst file. DB2 was engineered to automatically Bind the utilities that you call if that utility that not been bound yet. It does this on the background and it is transparent to the user. A bind on the utilities has to be performed every time you install a FixPak or when you connect to new host machine.

You can Export a DB2 table in distributed platform into an IXF file then Import this file to an existing table. Only IXF file format is supported. Only the Import INSERT option can be used. The Import REPLACE and the Import CREATE option are not allowed. You need to have DB2 Connect running in your system. This however is automatically installed with DB2 Enterprise Edition or DB2 Enterprise Extended Edition.

Example on moving data from distributed to z/OS using Import - Export

This example will move the DEPARTMENT table in the SAMPAIX8 database from a distributed DB2 database to an existing table PAOLOR2.DEPT in a DB2 for z/OS database.

Step 1. Connect to source database

Connect to the source database.

From the DB2 command prompt enter:

```
CONNECT TO SAMPAIX8
```

Step 2. Export the table

This step will export the DEPARTMENT table to an IXF file called DEPTFILE.IXF. The table information and the data will be in this file. You have to include the schema name in the SELECT statement if you are not the owner of the table:

```
EXPORT TO DEPTFILE.IXF OF IXF SELECT * FROM DB2INST2.DEPARTMENT
```

If the export is successful you will see a message on the screen indicating the number of rows exported and the file name where the data is stored.

Step 3. Connect to the target database

Connect to the target mainframe database. Notice that to connect to the mainframe database need to specify the DB2 subsystem name in the connect statement. In our example DB2G is the subsystem name in the DB2 for z/OS. You will be prompted for a password after entering this command:

```
CONNECT TO DB2G USER MYUSERNM
```

Step 4. Import the IXF file into the source table

To import the IXF file that you created in step 2, enter the command below. When you are using the Import tool in moving data from distributed to mainframe, you can only use the

INSERT option of the Import tool. Take note also that only the IXF file format can be used in this scenario:

```
IMPORT FROM DEPTFILE.IXF OF IXF MESSAGES MESSAGE.TXT INSERT INTO PAOLOR2.DEPT
```

If the import is successful you should see the message on the number of rows that was inserted into the table. The messages generated from this command will be placed in the text file MESSAGE.TXT

This procedure applies also to moving data with LOB columns. You export the table with the LOBs to an IXF file then import it using the same steps we outlined above.

Using pipes in export-import to move data from distributed to z/OS

The export-import tools can also use pipes in moving data from distributed to z/OS. Before you execute the export-import commands you need to create the pipe in AIX using the command:

```
mkfifo pipename
```

or

```
mknod pipename p
```

When the pipe is already created treat it like a regular IXF file. Follow the import-export steps 1 to 4 outlined above. The LOBSINFILE option cannot be used with pipes.

10.2.4 SQL Insert with subselect in a Federated Database

You can use an SQL Insert and a subselect statement to move data from distributed platform to mainframe platform. The prerequisite of this method is a Federated Database set-up and a DB2 UDB Version 8. With DB2 Version 8 you can already retrieve and update data in a Federated Database; in previous versions you can only read data from it. To use a Federated Database in moving data, you need to assign nicknames for the source and the target database tables. The steps in creating a Federated Database are outlined in Appendix A.

After the Federated Database is set-up, you can use the SQL Insert to write data to the target z/OS database table. The SQL subselect is used to retrieve the data from the source DB2 on distributed database table.

Example of using the SQL Insert with subselect in a Federated Database

You first need to set-up the Federated Database creating a nickname on the DB2 for z/OS table. In this example, our target table is the PAOLOR7.DEPT table in the z/OS database which is nicknamed as DB2ADMIN.DEPT390T in the Federated Database. Our source table is the DEPARTMENT table in the client machine. DB2INST2 is the schema name of the DEPARTMENT table.

In the DB2 CLP in the client machine, issue these commands:

First, connect to the database in the client machine where the Federated Database is defined:

```
CONNECT TO SAMPAIX8
```

Then, issue the Insert with subselect command to transfer the data to the target:

```
INSERT INTO DB2ADMIN.DEPT390T SELECT * FROM DB2INST2.DEPARTMENT
```

You can also specify the columns that you want to retrieve in your Select statement. The subselect is like a regular SQL Select statement. You can have a Where clause and scalar functions in your Select statement as long as the result set matches the columns of your target table.

10.3 Moving data between two DB2 for z/OS databases

There are five sets of tools/utilities that we will use in moving data from one DB2 for z/OS database to another. These are:

- DB2 Cross Loader
- DB2 Data Propagator
- DB2 Unload and Load
- HPU for z/OS and Load
- Insert with subselect

Database objects and environment

The database objects and environments used in this section are listed in Table 10-4. Most of the examples will be based on this table unless the example states otherwise.

Table 10-4 Database objects in moving data across DB2 for z/OS databases

Objects	Object name
Source DB2 for z/OS	
Subsystem name	DB2G
Table space name	DSNDB04.DEPT
Schema name	PAOLOR7
Table name	DEPT
Columns	DEPTNO CHAR(3) NOT NULL, DEPTNAME VARCHAR(36) NOT NULL, MGRNO CHAR(6), ADMRDEPT CHAR(3) NOT NULL, LOCATION CHAR(16)
Target DB2 for z/OS	
Subsystem name	DB2G
Table space name	DSNDB04.DEPTCOPY
Table name	PAOLOR2
Table name	DEPT
Columns	DEPTNO CHAR(3) NOT NULL, DEPTNAME VARCHAR(36) NOT NULL, MGRNO CHAR(6), ADMRDEPT CHAR(3) NOT NULL, LOCATION CHAR(16)

10.3.1 Using Cross Loader to move data from/to DB2 for z/OS

Cross Loader can also move data between two DB2 tables residing in z/OS.

Example of using DB2 Cross Loader to move data in DB2 in z/OS

You need DSNUTILB to be APF authorized. In this example, this will insert new rows into the PAOLOR2.DEPT table and retain the old ones. If you want to replace the existing rows in the table, you need to specify REPLACE instead of RESUME YES. The source of the data is the PAOLOR7.DEPT table. See Example 10-2.

The Load option used in this example is explained in Table 10-5.

Table 10-5 Options used in Load

Option	Description
RESUME YES	The new rows from PAOLOR7.DEPT will be appended to the existing rows of PAOLOR2.DEPT. Existing rows in the PAOLOR2.DEPT table will remain in the table. Other Option: REPLACE, RESUME NO

Look at Example 10-2 to see an example of a DB2 Cross Loader job for moving data between two mainframe DB2 tables.

Example 10-2 Cross Loader sample JCL moving from z/OS to z/OS

```
//PAOLCRLO JOB  (999,POK),'DB2CRLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* CROSS LOADER                                                  *
//*****************************************************************
//*
//EXEC     EXEC PGM=DSNUTILB,PARM='DB2G,CRLOAD'
//STEPLIB  DD DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUT1   DD DSN=PAOLOR3.LOAD.STEP1.SYSUT1,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),RLSE,,ROUND)
//SORTOUT  DD DSN=PAOLOR3.LOAD.STEP1.SORTOUT,
//         DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),RLSE,,ROUND)
//SYSIN    DD *
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT DEPTNO, DEPTNAME, MGRNO, LOCATION
    FROM PAOLOR7.DEPT
  ENDEXEC
  LOAD DATA
    INCURSOR C1
    RESUME YES
    INTO TABLE PAOLOR2.DEPT
//*
```

10.3.2 Data Propagator

It is also possible to move data from one z/OS database to another using DB2 Data Propagator. Read 11.5, “Data Propagator” on page 279.

10.3.3 Unload and Load

The DB2 Unload utility and the Load utility are two of the most common tools used to move data from z/OS to z/OS databases. The Unload utility produces an output file compatible to the Load utility input file format. The Load utility can load one or more tables of a table space. It builds or extends any indexes defined on the table. If the table space already contains data, you can choose whether you want to add the new data to the existing data (RESUME YES option) or replace (REPLACE option) the existing data.

Example of moving data within z/OS using DB2 Unload and Load

The first JCL step of this job is unloading data from the z/OS DB2 table. The source table PAOLOR7.DEPT is unloaded to a sequential data set PAOLOR2.MD.UNLOAD.DATAVAR. The varchar columns are not padded (NOPAD option). The second JCL step loads the output file created by the first step and inserts it to the target table. The existing rows in the table are retained (RESUME YES option) and the new rows are appended by the Load utility.

In the EXEC statement of the JCL, the UID (utility ID) of the first job step should be 'UNLOAD' and the UID of the second step is LOAD. The parameter 'SYSTEM' is the DB2 subsystem name.

The SORTOUT data set in the LOAD step can be left blank because there is no index created in the PAOLOR7.DEPT table during the LOAD. The data set PAOLOR2.MD.UNLOAD.DATAVAR is the output data set in the Unload utility job step and it will be the input data set in the Load utility job step. See Example 10-3.

The database object used in this example are in the table below. See Table 10-4 on page 251. The Load and Unload option used in this example are explained below. See Example 10-6.

Table 10-6 Options used in Example 10-3

Option	Description
Unload Options	
NOPAD	VARCHAR fields that have not consumed the entire allocated space will NOT be padded.
Load Options	
RESUME YES	The new rows will be appended to the existing row. Existing rows in the table will be remain in the table. Other Option: REPLACE, RESUME NO
EBCDIC	Data will be loaded in EBCDIC format Other options: ASCII, ASIS
LOG NO	The Load activity will not be logged. Other option: LOG YES
CCSID(00037,00000,00000)	CCSID value for USA, Canada, Australian English code page. This is the default value. You only need to specify this if you are going to use another code page. Other options: CCSID(00000, 05035,00000) for Japanese; CCSID(00000,01371.00000) for Chinese CCSID(00865,00000,00000) for Denmark, Sweden, Norway, Finland

Look at Example 10-3 to see an example of a DB2 Unload and Load utility job for moving data between two mainframe DB2 tables.

Example 10-3 JCL to Unload and Load data to and from DB2 for z/OS tables

```
//PAOLLOAD JOB  (999,POK),'DB2LOAD-1',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//JOBLIB   DD DSN=DB2G7.SDSNLOAD,DISP=SHR
```

```
//*
//*****************************************************************
//* STEP 1: DB2 UNLOAD UTILITY NOPAD (VARIABLE FORMAT)          *
//*****************************************************************
//*
//UNLOAD   EXEC DSNUPROC,UID='UNLOAD',UTPROC='',SYSTEM='DB2G'
//SYSREC   DD DSN=PAOLOR2.MD.UNLOAD.DATAVAR,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(2,1),RLSE)
//SYSPUNCH DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  UNLOAD DATA NOPAD
  FROM TABLE PAOLOR7.DEPT
//*
//*****************************************************************
//* STEP 2: DB2 LOAD UTILITY - LOADS DATA UNLOADED IN STEP 1    *
//*****************************************************************
//*
//LOAD     EXEC DSNUPROC,UID='LOAD',UTPROC='',SYSTEM='DB2G'
//SYSREC   DD DSN=PAOLOR2.MD.UNLOAD.DATAVAR,DISP=SHR
//SYSUT1   DD SYSOUT=*
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "PAOLOR2 "."DEPT              "
 WHEN(00001:00002 = X'005C')
 ( "DEPTNO            " POSITION(  00003:00005) CHAR(003)
 , "DEPTNAME          " POSITION(  00006)       VARCHAR
 ,  DSN_NULL_IND_00003  POSITION(  *)           CHAR(1)
 , "MGRNO             " POSITION(  *)           CHAR(006)
                          NULLIF(DSN_NULL_IND_00003)=X'FF'
 , "ADMRDEPT          " POSITION(  *)           CHAR(003)
 ,  DSN_NULL_IND_00005  POSITION(  *)           CHAR(1)
 , "LOCATION          " POSITION(  *)           CHAR(016)
                          NULLIF(DSN_NULL_IND_00005)=X'FF'
 )
//*
```

Suggested tasks after performing Load

These are some tasks that should be done after the LOAD operation. Not all of them are necessary. It depends on the status of your database after the LOAD operation was performed.

- CHECK DATA if referential integrity is violated

  LOAD places a table space in the CHECK-pending status if its referential integrity is in doubt, or its check constraints are violated. The intent of the restriction is to encourage the use of the CHECK DATA utility. That utility locates invalid data, and optionally, removes it. If it removes the invalid data, the data remaining satisfies all check and referential constraints, and the CHECK-pending restriction is lifted.

- Reorganizing an auxiliary index after LOAD

  Indexes on the auxiliary tables are not built during the BUILD phase. Instead, LOB values are inserted (not loaded) into auxiliary tables during the RELOAD phase as each row is loaded into the base table, and each index on the auxiliary table is updated as part of the INSERT operation. Because the LOAD utility inserts keys into an auxiliary index, free space within the index might be consumed and index page splits might occur. Consider

reorganizing an index on the auxiliary table after LOAD completes to introduce free space into the index for future INSERTs and LOADs.

- Copying the loaded table space or partition

  If you have used LOG YES, consider taking a full image copy of the loaded tablespace or partition to reduce the processing time of subsequent recovery operations. If you also specified RESUME NO or REPLACE, indicating that this is the first load into the table space, we recommend that you take two or more full image copies to enable recovery. Alternatively, we recommend that you take primary and backup inline copies when you do a LOAD REPLACE; full table space or partition image copies taken after the LOAD completes are not necessary.

- Re-bind application plans that depend on the loaded table

  Use either the STATISTICS option to collect inline statistics, or the RUNSTATS utility so that the DB2 catalog statistics take into account the newly loaded data, and DB2 can select SQL paths with accurate information. Following this, rebind any application plans that depend on the loaded tables to update the path selection of any embedded SQL statements.

- Removing COPY-pending status

  If you Load with LOG NO and do not take an inline copy, LOAD places a table space in the COPY-pending status. Immediately after that operation, DB2 cannot recover the table space (though you can, by loading it again). Prepare for recovery, and turn off the restriction, by making a full image copy using SHRLEVEL REFERENCE. (If you end the copy job before it is finished, the table space is still in COPY-pending status.)

  You can also remove the restriction by one of these operations:

  - LOAD REPLACE LOG YES
  - LOAD REPLACE LOG NO with an inline copy
  - REORG LOG YES
  - REORG LOG NO with an inline copy
  - REPAIR SET with NOCOPYPEND

  If you use LOG YES and do not make an image copy of the table space, subsequent recovery operations are possible, but will take longer than if you had made an image copy.

  A table space in COPY-pending status can be read without restriction; however, it cannot be updated.

10.3.4 HPU for z/OS and Load

The HPU for z/OS can be used to unload a DB2 table. The output data set can be formatted in different ways, so that it can be compatible to the DB2 Load utility. The HPU can produce an output with DSNTIAUL format, USER format (positional ASCII), DELIMITED ASCII format, and VARIABLE format.

In moving data between two DB2 for z/OS tables, you can use three output formats compatible with the Load utility. These are: DSNTIAUL, VARIABLE format, and the USER format. Of these three, it is best to use the DSNTIAUL if you are not doing data type transformation (example decimal to integer, or char to varchar) while unloading data. It is a lot easier and less prone to error to use the DSNTIAUL. However, if you need to do some data type transformations, it is best to use the USER format.

Example 1: HPU and Load to move data in DB2 in z/OS tables

In this example, we are going to move data from a DB2 table residing in z/OS to another DB2 table in z/OS. The source table is PAOLOR7.DEPT and the target table is PAOLOR2.DEPT. We are going to use the DSNTIAUL format because it is one of the formats compatible with

the LOAD utility. Notice that DB2 is set to YES. This is so that if the SQL statement in the Unload job cannot be executed by HPU, the job can be passed to DB2 for retrieval of rows.

You should also specify QUIESCE YES and QUIESCECAT YES so that any late changes in the source table will be included in the output. When this is specified, you can only proceed with it when the table is finished in the QUIESCE operation, and the data in the buffer is already flushed in the disk.

The SORTOUT and SYSUT1 data sets are not required because the target table in this example is not indexed. See Example 10-4.

The database object used in this example are in Table 10-4 on page 251. The options used in this example are explained in Table 10-7.

Table 10-7 Options used in Example 10-4

Option	Description
Unload options	
DB2 YES	HPU can pass the query to DB2 for execution if the SQL Select cannot be performed by the HPU program. Other options: DB2 NO, DB2 FORCE
QUIESCE YES	QUIESCE will be done on the table space before unloading it. Other option: QUIESCE NO
QUIESCECAT YES	QUIESCE will be done on the DB2 catalog's table spaces before unloading. Other option: QUIESCECAT NO
DATE DATE_A	Date format is MM-DD-YYYY
Load options	
DSNTIAUL	Format of output will be same as DSNTIAUL program output.
LIKE PAOLOR2.DEPT	Indicates that HPU uses the characteristics of the table model PAOLOR2.DEPT as parameters, and formats the data set to allow the LOAD of this table.
REPLACE	The new rows from PAOLOR7.DEPT will be appended to the existing rows of PAOLOR2.DEPT. Existing rows in the PAOLOR2.DEPT table will remain in the table. Other Options: RESUME YES, RESUME NO
EBCDIC	Refer to Table 10-6 on page 253
LOG YES	Refer to Table 10-6 on page 253
CCSID(0037)	Refer to Table 10-6 on page 253

Look at Example 10-4 to see an example of a HPU job for moving data between two mainframe DB2 tables

Example 10-4 JCL for HPU and Load

```
//PAOLJOB2 JOB  (999,POK),'HPUNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//* STEP 1: HP UNLOAD TOOL - UNLOAD IN FORMAT DSNTIAUL         *
//*************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,HPUNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSIN    DD  *
     UNLOAD TABLESPACE
      DB2 YES
      QUIESCE YES QUIESCECAT YES
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1)
         FORMAT DSNTIAUL LIKE PAOLOR2.DEPT
         LOADDDN LOADDD LOADOPT (LOG YES REPLACE)
/*
//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//LOADDD   DD  SYSOUT=*
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATA1,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
//*
//*************************************************************
//* STEP 2: LOAD UTILITY - LOADS DATA UNLOADED IN STEP 1       *
//*               (FORMAT: DSNTIAUL)                           *
//*************************************************************
//*
//STEP2    EXEC DSNUPROC,UID='LOADUTIL',UTPROC='',SYSTEM='DB2G'
//STEPLIB  DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSREC   DD DSN=PAOLOR2.MD.HPUNLOAD.DATA1,DISP=SHR
//SYSUT1   DD SYSOUT=*
//SORTOUT  DD SYSOUT=*
//SYSIN    DD *
LOAD DATA LOG YES REPLACE
EBCDIC CCSID(0037)
INTO TABLE PAOLOR2.DEPT
(
DEPTNO
     POSITION (     1  )         CHAR (     3 ),
DEPTNAME
     POSITION (     4  )         VARCHAR,
MGRNO
     POSITION (    42  )         CHAR (     6 )
     NULLIF(    48 ) = '?',
ADMRDEPT
     POSITION (    49  )         CHAR (     3 ),
LOCATION
     POSITION (    52  )         CHAR (    16 )
```

```
     NULLIF(    68 ) = '?'
)
//*
```

Example 2: HPU and Load with the last full image copy as source

The HPU can unload the last incremental copy of a database back-up and place it in a data set. This data set will then be used as input of the Load utility. In this example, we have two JCL steps in our job. The first step unloads the the full image copy and the second step loads to the target table. The latest full image copy is automatically updated in the SYSIBM.SYSCOPY table every time a back-up is made on a table space. HPU refers to the SYSCOPY table to determine where the latest full image copy is stored. The output data set of the HPU job step is the PAOLOR2.MD.HPUNLOAD.DATA2, which has a DSNTIAUL format.

In the second JCL step, we use the LOAD utility to load data into the PAOLOR3.DEPT table. The SYSUT1 and the SORTOUT data sets are needed because the PAOLOR3.DEPT is an indexed table. The index will be re-written after the new data is inserted into the table. Hence, the SORTOUT and SYSUT1 data sets will be used as work data sets for the sort and indexing. See Example 10-5.

Database objects and options

The database object used in this example are in Table 10-4 on page 251. The Unload and Load options used in this example are explained below. See Table 10-8.

Table 10-8 Options used in Example 10-5

Option	Description
Unload options	
COPYDDN LAST_IC	This option indicates that the last image copy of PAOLOR2.DEPT will be used as the source of data. This is different from the existing PAOLOR2.DEPT table. The last image copy is the latest back-up of the table. The name of the data set that has last image copy will be read from the SYSIBM.SYSCOPY table.
DB2 YES	Refer to Table 10-7 on page 256
QUIESCE YES	Refer to Table 10-7 on page 256
QUIESCECAT YES	Refer to Table 10-7 on page 256
DATE DATE_A	Refer to Table 10-7 on page 256
Load options	
DSNTIAUL	Refer to Table 10-7 on page 256
LIKE PAOLOR2.DEPT	Indicates that HPU uses the characteristics of the table model PAOLOR2.DEPT as parameters, and formats the data set to allow the LOAD of this table.

Option	Description
REPLACE	The new rows from PAOLOR2.DEPT will be appended to the existing rows of PAOLOR3.DEPT. Existing rows in the PAOLOR3.DEPT table will remain in the table. The rows will be sorted and indexed because PAOLOR3.DEPT is an indexed table. Hence you are required to specify the SORTOUT and SYSUT1 data sets.
EBCDIC	Refer to Table 10-6 on page 253
LOG YES	Refer to Table 10-7 on page 256
CCSID(0037)	Refer to Table 10-6 on page 253

Look at Example 10-5 to see a HPU job for moving data between two mainframe DB2 tables.

Example 10-5 HPU and Load using a full image copy of the back-up

```
//PAOLJOB3 JOB  (999,POK),'HPUNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* STEP 1: HPU TOOL - UNLOAD IN FORMAT DSNTIAUL          *
//*            (UNLOAD FROM LAST INCREMENTAL COPY)              *
//*****************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,HPUNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSIN    DD  *
     UNLOAD TABLESPACE DSNDB04.DEPT
      COPYDDN LAST_IC

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1)
         FORMAT DSNTIAUL LIKE PAOLOR2.DEPT
         LOADDDN LOADDD LOADOPT (LOG YES RESUME YES)

//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//LOADDD   DD  SYSOUT=*
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATA2,
//             DISP=(NEW,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
//*
//*****************************************************************
//* STEP 2: LOAD UTILITY - LOADS DATA UNLOADED IN STEP 1        *
//*                (FORMAT: DSNTIAUL)                           *
//*****************************************************************
//*
//STEP2    EXEC DSNUPROC,UID='LOADUTIL',UTPROC='',SYSTEM='DB2G'
//STEPLIB  DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSREC   DD DSN=PAOLOR2.MD.HPUNLOAD.DATA2,DISP=SHR
//SYSUT1   DD  DSN=PAOLOR2.LOAD.SYSUT1,
```

```
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(1,5),RLSE)
//SORTOUT  DD  DSN=PAOLOR2.LOAD.SORTOUT,
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(1,5),RLSE)
//SYSIN    DD *
LOAD DATA LOG YES RESUME YES
EBCDIC CCSID(0037)
INTO TABLE PAOLOR2.DEPT
(
DEPTNO
     POSITION (     1  )         CHAR (     3 ),
DEPTNAME
     POSITION (     4  )         VARCHAR,
MGRNO
     POSITION (    42  )         CHAR (     6 )
     NULLIF(    48 ) = '?',
ADMRDEPT
     POSITION (    49  )         CHAR (     3 ),
LOCATION
     POSITION (    52  )         CHAR (    16 )
     NULLIF(    68 ) = '?'
)
//*
```

Example 3: HPU from two DB2 z/OS source tables to one target

In this example we unload data from two DB2 for z/OS tables and put it in one target DB2 for z/OS table. Notice that PAOLOR2.DEPT(table 1) and PAOLOR7.DEPT(table 2) are located in different table spaces. One JCL job will be used for the unload and load operation. See Example 10-6.

Database objects

The database object used in this example are in Table 10-4 on page 251. Another source table is used in this example. PAOLOR3.DEPT has similar properties as PAOLOR7.DEPT, but they are located in different table spaces.

The options used in this example are explained below. See Table 10-9.

Table 10-9 Options used in Example 10-6

Option	Description
Unload options	
DB2 NO	This option means DB2 cannot be called to retrieve the rows in case HPU cannot perform the query. In this example, there is no need to call DB2 because HPU can perform a simple Select statement.
QUIESCE YES	A QUIESCE will be done on table spaces DSNDB04.DEPT1ZN9 and DSNDB04.DEPT1WD before it is unloaded. This will make you sure that data in the buffer will be flushed to the disk before Unload is performed.
QUIESCECAT NO	QUIESCE will NOT be done on the DB2 catalog's table spaces before unloading.
DATE DATE_A	Refer to Table 10-7 on page 256

Option	Description
Load options	
DSNTIAUL	Refer to Table 10-7 on page 256
LIKE PAOLOR2.DEPT	Indicates that HPU uses the characteristics of the table model PAOLOR2.DEPT as parameters, and formats the data set to allow the LOAD of this table.
REPLACE	New rows from PAOLOR2.DEPT and PAOLOR7.DEPT will be appended to the existing rows of PAOLOR3.DEPT. Existing rows in the PAOLOR3.DEPT table will be remain in the table. The target table will be sorted and indexed after the LOAD. Hence, you are required to specify the SYSUT1 and SORTOUT data sets.
EBCDIC	Refer to Table 10-6 on page 253
LOG YES	Refer to Table 10-6 on page 253
CCSID(0037)	Refer to Table 10-6 on page 253

See Example 10-6, for a sample JCL to use HPU to unload two DB2 tables and load it to one table using the Load utility.

Example 10-6 JCL to HPU two DB2 for z/OS tables to one target table

```
//PAOLJOB4 JOB  (999,POK),'HPUNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* STEP 1: HP UNLOAD TOOL - UNLOAD IN FORMAT DSNTIAUL          *
//*****************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,HPUNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSIN    DD  *
     UNLOAD TABLESPACE
      DB2 NO
      QUIESCE YES QUIESCECAT NO
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1)
         FORMAT DSNTIAUL LIKE PAOLOR2.DEPT
         LOADDDN LOADDD1

       SELECT * FROM PAOLOR3.DEPT
         OUTDDN (UNLDDN2)
         FORMAT DSNTIAUL LIKE PAOLOR2.DEPT
         LOADDDN LOADDD2
/*
//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
```

```
//LOADDD1  DD  DUMMY
//LOADDD2  DD  DUMMY
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATAJO41,
//             DISP=(NEW,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//UNLDDN2  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATAJO42,
//             DISP=(NEW,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//*
//*************************************************************
//* STEP 2: LOAD UTILITY - LOADS DATA UNLOADED IN STEP 1       *
//*                (FORMAT: DSNTIAUL)                          *
//*************************************************************
//*
//STEP2    EXEC DSNUPROC,UID='DB2LOAD',UTPROC='',SYSTEM='DB2G'
//STEPLIB  DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSREC   DD DSN=PAOLOR2.MD.HPUNLOAD.DATAJO41,DISP=SHR
//         DD DSN=PAOLOR2.MD.HPUNLOAD.DATAJO42,DISP=SHR
//SYSUT1   DD  DSN=PAOLOR2.LOAD.SYSUT1,
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//SORTOUT  DD  DSN=PAOLOR2.LOAD.SORTOUT,
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//SYSIN    DD *
LOAD DATA LOG YES REPLACE
EBCDIC CCSID(0037)
INTO TABLE PAOLOR2.DEPT
(
DEPTNO
     POSITION (     1  )         CHAR (     3 ),
DEPTNAME
     POSITION (     4  )         VARCHAR,
MGRNO
     POSITION (    42  )         CHAR (     6 )
     NULLIF(    48 ) = '?',
ADMRDEPT
     POSITION (    49  )         CHAR (     3 ),
LOCATION
     POSITION (    52  )         CHAR (    16 )
     NULLIF(    68 ) = '?'
)
//*
```

Example 4: Move data across DB2 subsystems in another LPAR

In this scenario, we move data from two tables residing in one DB2 subsystem in LPAR DEV1 to one target table residing in another DB2 subsystem in LPAR PROD1. The pre-requisite is that you need to have at least a Select privilege on the source tables and an update privilege on the target table in the other subsystem.

We will use the same HPU and Load options as the one we used in Example 3. The only condition that we changed is that the target table resides in a different DB2 subsystem in a different LPAR.

We need to separate the two JCL steps and run the jobs in different LPARs. The job for the HPU should be submitted at the DEV1 LPAR, while the job for the Load utility should be submitted at the PROD1 LPAR.

The job in Example 10-7 will unload data from the source table in DB2G in LPAR DEV1. Hence, this job should be run at the LPAR DEV1.

Example 10-7 HPU job to extract data from DB2G subsystem

```
//PAOLJOB4 JOB  (999,POK),'HPUNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* STEP 1: HP UNLOAD TOOL - UNLOAD IN FORMAT DSNTIAUL          *
//*****************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,HPUNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSIN    DD  *
     UNLOAD TABLESPACE
      DB2 NO
      QUIESCE YES QUIESCECAT NO
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR7.DEPT
         OUTDDN (UNLDDN1)
         FORMAT DSNTIAUL
         LOADDDN LOADDD1

       SELECT * FROM PAOLOR3.DEPT
         OUTDDN (UNLDDN2)
         FORMAT DSNTIAUL
         LOADDDN LOADDD2
/*
//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//LOADDD1  DD  DUMMY
//LOADDD2  DD  DUMMY
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATAJO41,
//             DISP=(NEW,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//UNLDDN2  DD  DSN=PAOLOR2.MD.HPUNLOAD.DATAJO42,
//             DISP=(NEW,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//*
```

The output data sets of the HPU job which are PAOLOR2.MD.HPUNLOAD.DATAJO41 and PAOLOR2.MD.HPUNLOAD.DATAJO42 should be made accessible to the PROD1 LPAR. It could be through FTP or through removable media. Then you can use these data sets as the input of the Load utility JCL (SYSREC).

To change the subsystem, you just need to change the SYSTEM parameter on the EXEC statement of the JCL, and set it to the DB2 subsystem name that you want to access. See Example 10-8.

Example 10-8 Load utility to write data to the D7F1 subsystem

```
//PAOLJOB4 JOB  (999,POK),'HPUNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//* STEP 2: LOAD UTILITY - LOADS DATA UNLOADED IN STEP 1       *
//*               (FORMAT: DSNTIAUL)                           *
//*************************************************************
//*
//STEP2    EXEC DSNUPROC,UID='DB2LOAD',UTPROC='',SYSTEM='D7F1'
//STEPLIB  DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//SYSREC   DD DSN=PAOLOR2.MD.HPUNLOAD.DATAJ041,DISP=SHR
//         DD DSN=PAOLOR2.MD.HPUNLOAD.DATAJ042,DISP=SHR
//SYSUT1   DD  DSN=PAOLOR2.LOAD.SYSUT1,
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//SORTOUT  DD  DSN=PAOLOR2.LOAD.SORTOUT,
//             DISP=(MOD,CATLG,CATLG),UNIT=SYSDA,
//             SPACE=(TRK,(10,10),RLSE)
//SYSIN    DD *
LOAD DATA LOG YES REPLACE
EBCDIC CCSID(0037)
INTO TABLE PAOLOR2.DEPT
(
DEPTNO
     POSITION (     1  )         CHAR (     3 ),
DEPTNAME
     POSITION (     4  )         VARCHAR,
MGRNO
     POSITION (    42  )         CHAR (     6 )
     NULLIF(    48 ) = '?',
ADMRDEPT
     POSITION (    49  )         CHAR (     3 ),
LOCATION
     POSITION (    52  )         CHAR (    16 )
     NULLIF(    68 ) = '?'
)
//*
```

10.3.5 SQL Insert with subselect in a Federated Database

This is only applicable to DB2 Version 8. Data can be moved across two different DB2 for z/OS tables using SQL Insert and sub-select. In this scenario, you need to create a nickname on the Federated Database of the source and target DB2 for z/OS tables. When nicknames have been created you can use these nicknames in the SQL Insert and Subselect statement like regular DB2 table names.

Using SQL Insert with subselect in two z/OS databases

To use a Federated Database in moving data, you need to assign nicknames for the source and the target database tables. The steps in creating a Federated Database are outlined in Appendix A.

After the Federated Database is set-up, you can use the SQL Insert to write data to the target z/OS database table. The SQL sub-select is used to retrieve the data from the source DB2 on distributed database table.

In this example, we nicknamed the source PAOLOR7.DEPT table in the z/OS database as DBADMIN.DEPT390S. The target table PAOLOR2.DEPT was nicknamed as DB2ADMIN.DEPT390T.

In the DB2 CLP in the client machine, issue these commands:

First, connect to the database in the client machine where the Federated Database is defined:

```
CONNECT TO SAMPAIX8
```

Then, issue the Insert with subselect command to transfer the data to the target:

```
INSERT INTO DB2ADMIN.DEPT390T
SELECT * FROM DB2ADMIN.DEPT390S
```

You can also specify the columns that you want to retrieve in your Select statement. The subselect is like a regular SQL Select statement. You can have a Where clause and scalar functions in your Select statement as long as the result set matches the columns of your target table:

```
INSERT INTO DB2ADMIN.DEPT390T(DEPTNO, DEPTNAME, ADMRDEPT)
SELECT DEPTNO,DEPTNAME,ADMRDEPT
FROM DB2ADMIN.DEPT390S
WHERE DEPTNO IN ('A00','B00','C00')
```

You can use SQL Insert with sub-select to move data across different DB2 subsystems. These DB2 subsystems have to be defined in the Federated Database. Once nicknames have been created for the remote relational table, you can use these nicknames like regular table names in the SQL Insert and sub-select statement.

Note: You can use DB2 Data Joiner to access data from different relational databases and use the SQL Insert with sub-select to move data between different relational databases, like Oracle or MS SQL Server. With this technique you are only limited by the databases that you can connect to your Federated Database system.

10.4 Summary and conclusions

In this chapter, we discussed moving data to DB2 for z/OS database. We focused on some tools and utilities that move data from DB2 distributed and from DB2 on z/OS. We showed examples of moving data across mainframe and distributed platforms. And we mentioned the considerations that you should bear in mind when moving data to another platform.

The examples in this chapter show how to move actual data. It is assumed that all database objects exist on the target database. See Chapter 9., “Getting ready for moving data” on page 213.

10.4.1 From distributed

Currently, there is no combination of high-speed utilities or tools to move large amounts of data from distributed to mainframe. When HPU for distributed database can produce a positional format, it can be a powerful combination with FTP and the Load utility on the mainframe.

We recommend the Cross Loader for z/OS to read data from distributed databases or from nicknames in a Federated Database system. This is based on three-part names with the location defined in the CDB. The Load utility will reserve the data to be loaded through a cursor.

If you are moving data in a regular basis, Data Propagator is a very convenient and effective tool to use. You have the option to copy only the changes in the table, which saves you the time needed to copy the entire table every time you do an update. With this tool, you can schedule the movement of data, and it will automatically perform the scheduled run.

With the Import utility on distributed, you can do SQL Insert for IXF formatted data from a file or named pipe. For the time being IXF data has to be produced by the Export utility. At present, HPU for MP cannot produce IXF.

You can create a nickname on mainframe table in a Federated Database system. By using SQL Insert into a nickname, you can sub-select from a table or another nickname in the Federated Database system.

10.4.2 From mainframe

In moving data within mainframe DB2 databases, the two main tools that we used to unload data are the HPU for z/OS and the Unload utility.

HPU for z/OS is a very efficient tool in unloading data from DB2 for z/OS databases. Its strength is its ability to read the VSAM files directly. It also consumes less CPU processing than the Unload utility. However, they are nearly equal in speed performance. HPU can customize the output using the SQL Select and the USER format. Another advantage of HPU, is it can read incremental copies. The HPU is recommended for unloading data when the Select statement can be performed by the HPU and not by the DB2 engine. The HPU is separately priced tool, while the Unload utility is included in the DB2 for z/OS Utility suite package.

You should consider changing the existing solutions that use the DSNTIAUL program. The HPU can produce output formats compatible with the DSNTIAUL. But it saves on the overhead time required to pass the query to DB2 if the query is relatively simple. The DSNTIAUL always call DB2 to perform its query.

There is no alternative to the Load utility in the mainframe. This will load data from the data set produced by HPU or from the Unload utility. The Cross Loader in z/OS also uses the Load utility taking input from a cursor declared on a mainframe table.

With regards to the Data Propagator, the same considerations as from distributed is valid.

10.4.3 Miscellaneous

Code page conversion in our examples are done automatically using the HPU, the Unload utility or the Load utility.

Moving data to a DB2 for z/OS database is supported by several tools and utilities. These tools and utilities are constantly being improved through PTFs and FixPaks. You can get the latest list of PTF and FixPaks on the Web sites:

http://www.ibm.com/support/us/

http://www-3.ibm.com/software/data/db2/

11

Moving data to DB2 Distributed

In this chapter we discuss the alternatives available when moving data into DB2 UDB on distributed platforms such as UNIX and Windows. Source databases can reside on any member of the DB2 Family, or even on other relational database systems. In each case we comment on usability, performance, and explicit restrictions.

The following data movement methods are described:

- Cross Load into a distributed environment (in DB2 UDB Version 8 only)
- Export followed by a Load or Import into a distributed environment
- SQL Insert containing a SELECT clause into a distributed environment (in DB2 UDB Version 8 only)
- Data Propagator
- HPU for z/OS followed by Load or Import into a distributed environment
- Unload utility on z/OS followed by Load or Import into a distributed environment
- HPU for MP followed by Load or Import into a distributed environment

11.1 An overview

The next two pictures reported in Figure 11-1 and Figure 11-2 show the various functions of data movements, which we will present and give examples on in this chapter.

Data movement from any DB2 to DB2 distributed

Figure 11-1 provides a graphical summary representation of the function, which can be used when moving data from DB2 members residing either on the mainframe or distributed platform, to a DB2 member on the distributed platform.

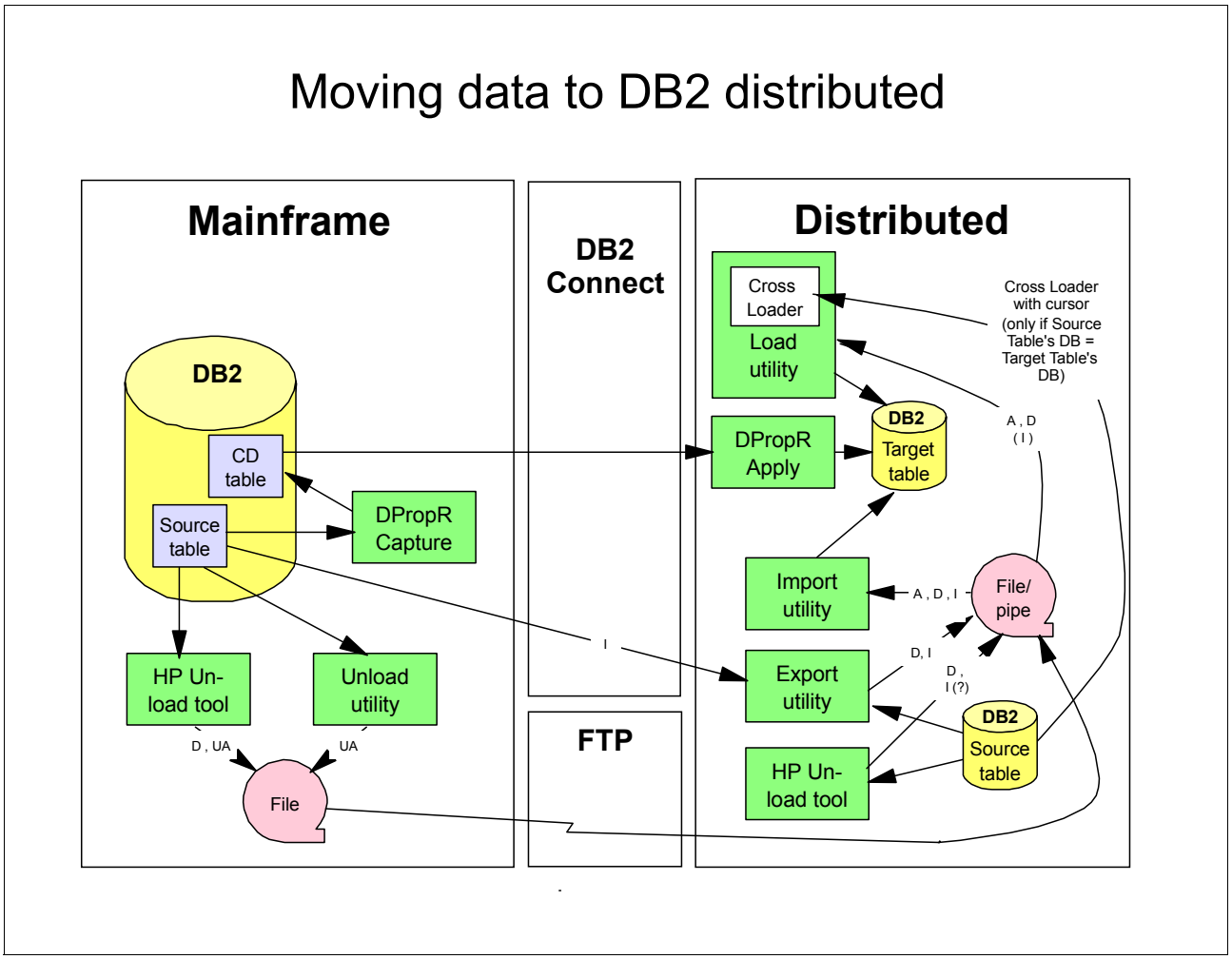

Figure 11-1 Moving data to distributed

The letters written on the arrows represents the formats that you can choose between:

A	Positional (fixed) format (ASC)
D	Delimited format (DEL)
I	Integrated exchange format, PC version (IXF)
(I)	The IXF format is accepted, but the table has to exist
UA	User defined Positional (fixed) format (ASC)

Data movement from any relational DB to DB2 distributed

Figure 11-2 shows how you can map an environment though a federated system on distributed by using nicknames. The source and the target tables can reside on any database that Federated Databases support. This can be within or outside the DB2 Family.

It is important to notice that the Export utility, the Cross Loader, and Insert with subselect can all read from a nickname, but only Insert with subselect can write to a nickname (from Version 8.)

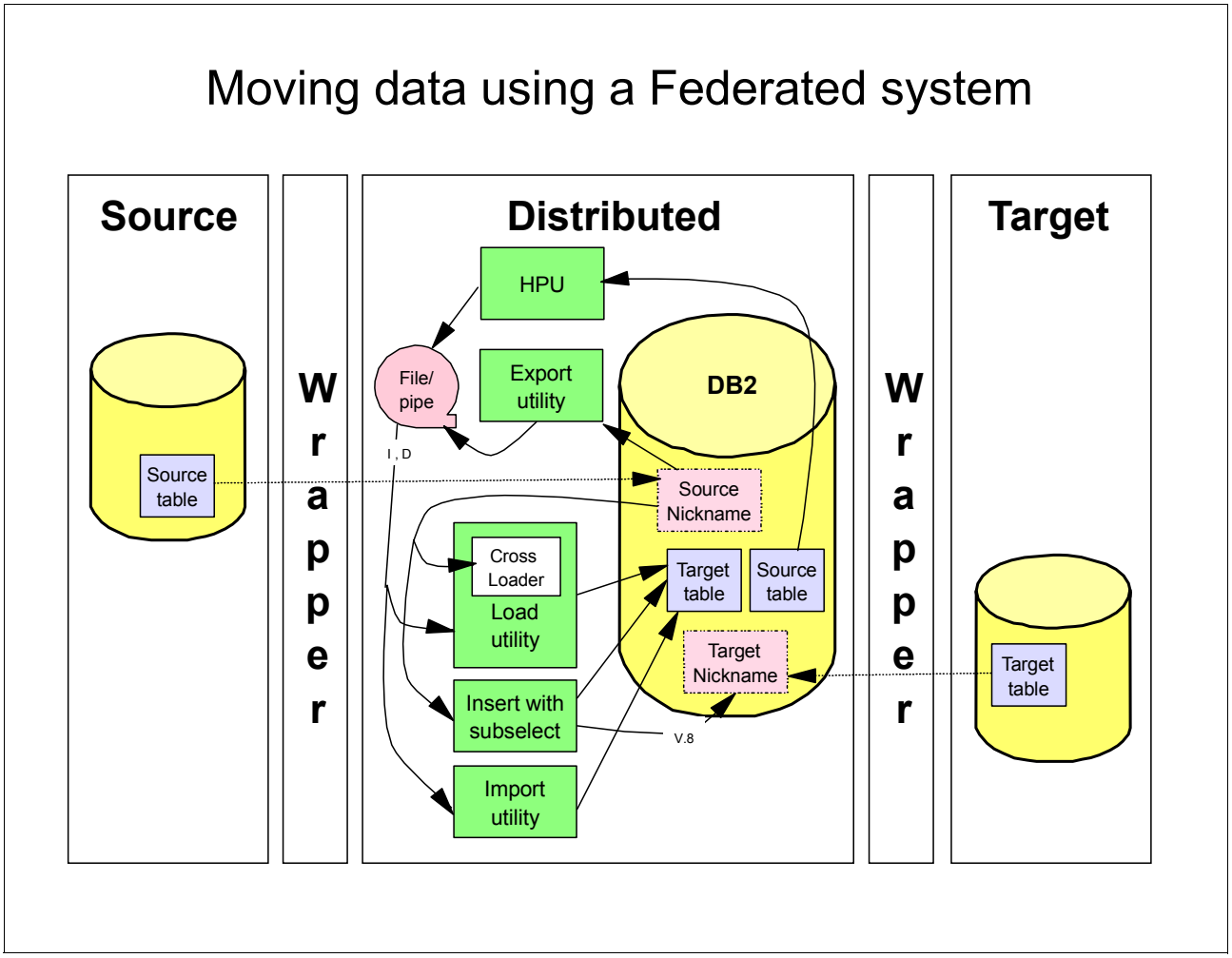

Figure 11-2 Moving data using a Federated System

The letters written on the arrows represents this information:

D	Delimited format (DEL)
I	Integrated exchange format, PC version (IXF)
V.8	DB2 UDB Version 8

11.1.1 File format considerations

When employing a data movement scheme where a file format must be chosen (DEL, ASC, or IXF), some considerations need to be made before making the choice. Each file format has its advantages and disadvantages, shown below:

- DEL (Delimited ASCII)

- Advantages
  - Unlimited record length
  - Data is human readable
- Disadvantages
  - This file format is not portable to a mainframe environment

> **Note:** If you are moving data from a Windows distributed environment to a UNIX distributed environment, keep in mind that the Export utility on Windows represents a newline as two characters (0x0D 0x0A). The Load utility and the Import utility in a UNIX environment both do not recognize this two character newline. They expect the newline character to be a single character (0x0A). Therefore, it will not load Windows exported data properly (for DEL format only, IXF is okay). Before performing the load or import, either FTP the DEL file from the source to target machine in ASCII mode (which will convert two character newlines to single character newlines), or perform the conversion yourself.

- ASC (positional ASCII):
  - Advantages
    - Data is human readable
    - Loading/Importing time is faster than DEL.
    - This file format is compatible with the Load utility in a mainframe environment
  - Disadvantages
    - Record length is limited to 32 KB.
    - Column length information is not encoded in the ASC format, therefore variable length character fields can only be loaded as variable length if the STRIPTBLANKS file type modifier is specified during Load or Import. STRIPTBLANKS strips trailing blanks encountered in character fields. However, doing this does not guarantee that data will be correct. Any character columns that originally contained trailing blanks will have their trailing blanks stripped when the data is loaded or imported.
    - Although this file format is compatible with the mainframe Load utility, it is not supported by the Import utility in a distributed environment when connected to a mainframe environment.

> **Note:** There are *two* types of positional ASCII: *FIXED* length and *FLEXIBLE* length. *FIXED* length means that the RECLEN option is specified during the import or load, and that all records are of a fixed length. *FLEXIBLE* length means that the RECLEN is *not* specified during the import or load, and records are delimited by the newline character.

- IXF (PC/IXF)
  - Advantages
    - Loading/Importing time is faster than DEL and ASC.
  - Disadvantages
    - Not supported in a partitioned database environment for Load, although it is supported for Import.
    - Data is in binary, and is not human readable.

- This file format is not portable to a mainframe environment, although it can be used through the Import utility on a distributed environment when a connection is made to a mainframe environment through DB2 Connect.

11.1.2 Index considerations

If your target table contains indexes, it may be a good idea to either defer index rebuilding or to drop the indexes and recreate them after the data bas been loaded or imported.

In the case where the *Import utility* is being used, SQL inserts are used to populate the target table. For tables containing indexes, this involves constant updates to the indexes. This may be less efficient than dropping the indexes before the import and recreating them afterwards, especially if the target table was originally empty or close to empty.

In the case where the *Load utility* is being used, the Load utility currently does not have the ability to build multiple indexes in parallel, whereas the SQL CREATE INDEX command does. Therefore, for a target table containing multiple indexes, it may be more efficient either to drop the indexes before the load and recreate them afterwards, or perform the load with INDEXING MODE DEFERRED, especially if the target table was originally empty or close to empty.

When a load to a table is executed with INDEXING MODE DEFERRED, the table's indexes are marked bad and the load's BUILD phase is skipped. Subsequent access of the table's indexes (apart from another load) will automatically force the indexes to be rebuilt.

11.1.3 Environment used for data movement examples

For the examples given in this section, we use a distributed database named *distdb* as the target database, and a mainframe database named *db2g* and a distributed database named *distdb2* as the source databases. *distdb* is a Federated Database on a federated system. *db2g* is locally CATALOGed as *hostdb* on the same system as distdb. **distdb** is locally CATALOGed as *distdb* on the same system as distdb2. For more details on defining a Federated Database, see Appendix A, "Defining a Federated Database" on page 299.

Data movement to distributed: description of db2g database

The source database, *db2g*, has the following characteristics:

- **db2g** resides on a mainframe machine at IP address 192.168.0.1, which can be accessed through port 12345.
- **db2g** can be accessed through the userid **paolor2** and password **xxyyzz**.

db2g contains the following tables:

PAOLOR2.EMP

COLUMN NAME	DATA TYPE	LENGTH	NULLS
EMPNO	CHAR	6	NO
FIRSTNME	VARCHAR	12	NO
MIDINIT	CHAR	1	NO
LASTNAME	VARCHAR	15	NO
WORKDEPT	CHAR	3	YES
PHONENO	CHAR	4	YES
HIREDATE	DATE		YES
JOB	CHAR	8	YES
EDLEVEL	SMALLINT		YES
SEX	CHAR	1	YES
BIRTHDATE	DATE		YES
SALARY	DECIMAL	(9, 2)	YES

```
BONUS              DECIMAL         ( 9, 2)  YES
COMM               DECIMAL         ( 9, 2)  YES
```

SC246300.LITERATURE

```
COLUMN NAME        DATA TYPE       LENGTH   NULLS
TITLE              CHAR            25       YES
IDCOL              ROWID                    NO
MOVLENGTH          INTEGER                  YES
LOBMOVIE           BLOB            2048     YES
LOBBOOK            CLOB            10240    YES
```

Data movement to distributed: description of hostdb database

db2g was CATALOGed locally as *hostdb* on the distributed system where *distdb* resides, by executing the following CLP commands:

```
CATALOG TCPIP NODE hostnode REMOTE 192.168.0.1 SERVER 12345
CATALOG DCS DATABASE rmtdbdcs AS db2g
CATALOG DATABASE rmtdbdcs AS hostdb AT NODE hostnode AUTHENTICATION DCS
```

Data movement to distributed: description of distdb database

The target database, *distdb*, has the following characteristics:

- **distdb** resides on a distributed system at IP address 192.167.1.1, which can be accessed through port 56789
- **distdb** is a Federated Database on a DB2 UDB Version 8 federated system.
- **distdb** can be accessed through the userid **markomp** and password **aabbcc**.

distdb contains the following tables:

MARKOMP.EMP

```
Column                         Type      Type
name                           schema    name                Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- ------
EMPNO                          SYSIBM    CHARACTER                6     0 No
FIRSTNME                       SYSIBM    VARCHAR                 12     0 No
MIDINIT                        SYSIBM    CHARACTER                1     0 No
LASTNAME                       SYSIBM    VARCHAR                 15     0 No
WORKDEPT                       SYSIBM    CHARACTER                3     0 Yes
PHONENO                        SYSIBM    CHARACTER                4     0 Yes
HIREDATE                       SYSIBM    DATE                     4     0 Yes
JOB                            SYSIBM    CHARACTER                8     0 Yes
EDLEVEL                        SYSIBM    SMALLINT                 2     0 Yes
SEX                            SYSIBM    CHARACTER                1     0 Yes
BIRTHDATE                      SYSIBM    DATE                     4     0 Yes
SALARY                         SYSIBM    DECIMAL                  9     2 Yes
BONUS                          SYSIBM    DECIMAL                  9     2 Yes
COMM                           SYSIBM    DECIMAL                  9     2 Yes
```

MARKOMP.LIT

```
Column                         Type      Type
name                           schema    name                Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- ------
TITLE                          SYSIBM    CHARACTER               25     0 Yes
IDCOL                          SYSIBM    VARCHAR                 40     0 No
MOVLENGTH                      SYSIBM    INTEGER                  4     0 Yes
LOBMOVIE                       SYSIBM    BLOB                  2048     0 Yes
LOBBOOK                        SYSIBM    CLOB                 10240     0 Yes
```

- **distdb** contains nicknames *MARKOMP.EMPHOST* and *MARKOMP.LITHOST* that refer to the tables in database *db2g PAOLOR2.EMP* and *SC246300.LITERATURE*, respectively. They were created using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
CREATE WRAPPER DRDA
CREATE SERVER hostsrvr TYPE DB2/MVS VERSION 7.0.0 WRAPPER DRDA AUTHORIZATION
"paolor2" PASSWORD "xxyyzz" OPTIONS (NODE 'hostnode', DBNAME 'HOSTDB')
CREATE USER MAPPING FOR markomp SERVER hostsrvr OPTIONS (REMOTE_AUTHID 'paolor2',
REMOTE_PASSWORD 'xxyyzz')
CREATE NICKNAME MARKOMP.EMPHOST FOR hostsrvr.PAOLOR2.EMP
CREATE NICKNAME MARKOMP.LITHOST FOR hostsrvr.SC246300.LITERATURE
```

Important: As mentioned above, distdb resides on a Version 8 federated system. Had distdb resided on a Version 7 federated system, the DBNAME value of the CREATE SERVER command would have specified the DB2G database directly instead of the locally CATALOGed database HOSTDB. For more details, please refer to the steps that describe the CREATE SERVER command in Appendix A, “Defining a Federated Database” on page 299.

Data movement to distributed: description of distdb2 database

The source database, *distdb2*, has the following characteristics:

- **distdb2** is a database on a different DB2 UDB Version 8 system.
- **distdb2** can be accessed through the userid **mikemp** and password **nnoopp**.

distdb2 contains the following table:

MIKEMP.EMP

Column name	Type schema	Type name	Length	Scale	Nulls
EMPNO	SYSIBM	CHARACTER	6	0	No
FIRSTNME	SYSIBM	VARCHAR	12	0	No
MIDINIT	SYSIBM	CHARACTER	1	0	No
LASTNAME	SYSIBM	VARCHAR	15	0	No
WORKDEPT	SYSIBM	CHARACTER	3	0	Yes
PHONENO	SYSIBM	CHARACTER	4	0	Yes
HIREDATE	SYSIBM	DATE	4	0	Yes
JOB	SYSIBM	CHARACTER	8	0	Yes
EDLEVEL	SYSIBM	SMALLINT	2	0	Yes
SEX	SYSIBM	CHARACTER	1	0	Yes
BIRTHDATE	SYSIBM	DATE	4	0	Yes
SALARY	SYSIBM	DECIMAL	9	2	Yes
BONUS	SYSIBM	DECIMAL	9	2	Yes
COMM	SYSIBM	DECIMAL	9	2	Yes

Data movement to distributed: description of distdb database

distdb was CATALOGed locally as *distdb* on the distributed system where *distdb2* resides, by executing the following CLP commands:

```
CATALOG TCPIP NODE dnode REMOTE 192.167.1.1 SERVER 56789
CATALOG DATABASE distdb AT NODE dnode
```

11.1.4 Graphical representation of the environment used in the examples

Figure 11-3 shows the example environment described in 11.1.3, “Environment used for data movement examples” on page 271. We have used three different databases:

- **db2g** is a mainframe system used as a source system.
- **distdb2** is a distributed system used as a source system.
- **distdb** is a federated system used as the target system.

Observe that:

- **db2g** is cataloged as **hostdb** on **distdb** and can be accessed from this.
- **distdb** is cataloged as **distdb** on **distdb2** and can be accessed from this.

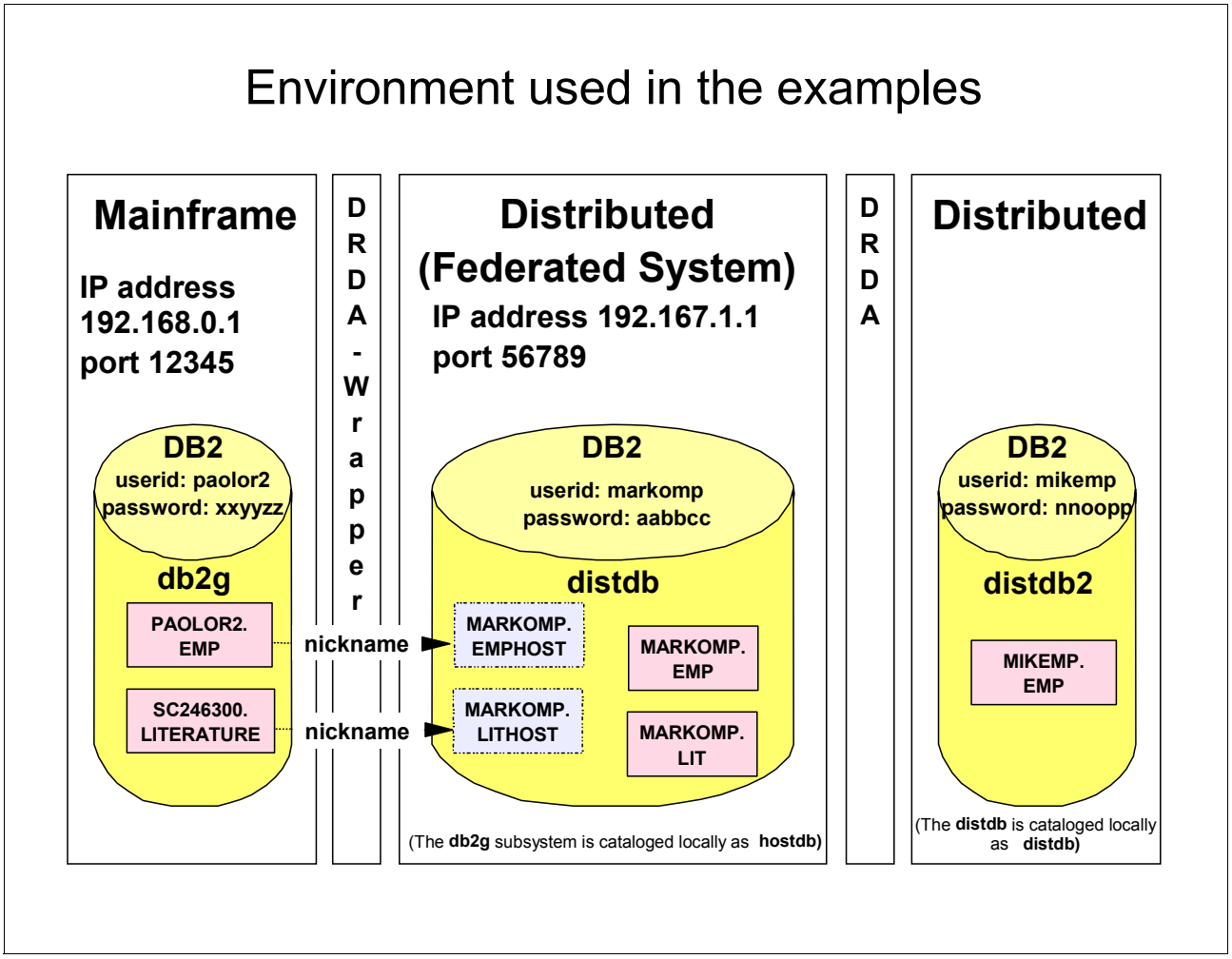

Figure 11-3 Environment for our examples

11.2 Cross loading

This method of data movement is applicable to any source server type that Federated Databases support.

Outline

1. If the source does not exist in the same database as the target, use Federated Database support to create a nickname for the source (see Appendix A, “Defining a Federated

Database" on page 299 for more details). Cross Load requires that both source and target table are accessible from the same database connection.

2. If performing the cross load through CLP, declare a cursor for selecting data from the source. This cursor will be used in the CLP load command that performs the cross load. Note that if the cross load is being invoked through the load API db2Load(), the SQL query is passed directly to the API and an intermediate declared cursor is not necessary.
3. Perform the cross load by invoking the load utility with the necessary options.

Evaluation

- Restrictions:
  - DB2 UDB V8 has to be used on the distributed environment.
  - A Federated database must be used if the source is not in the same database as the target.
  - For the Load, see Chapter 6, "Load with DB2 Distributed" on page 91.
- Performance:
  - Using the fastest tool available on the target side
  - Data retrieved from the source through an SQL query statement
- Usability:
  - The process is administered only on a distributed environment.
  - Since the source can be a nickname, the source can be anything that Federated databases support, even other database products. The only downside is that some additional setup is needed to create table nicknames when needed.
  - It is easy to use, and no intermediate data files are produced.

Example

The following example demonstrates how the Cross Loader can be used to move data from a mainframe system to distributed. However, it can be used to move data from *any* source that Federated Databases support, to distributed.

Moving data from PAOLOR2.EMP to MARKOMP.EMP

To move data from the host table PAOLOR2.EMP to the distributed table MARKOMP.EMP (through the nickname MARKOMP.EMPHOST), you can use the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
DECLARE hostcursor CURSOR FOR select * from MARKOMP.EMPHOST
LOAD FROM hostcursor OF cursor MESSAGES load.msg REPLACE INTO MARKOMP.EMP
```

In this example:

- In the Federated Database distdb, MARKOMP.EMPHOST was set up as a nickname for the table PAOLOR2.EMP that belonged to the mainframe database db2g (see 11.1.3, "Environment used for data movement examples" on page 271 for more details.)
- This example is applicable to both a partitioned database environment and a non-partitioned database environment.

11.3 Export followed by Load or Import

This method of data movement is applicable when moving data from a distributed environment, an environment supported by DB2 Connect (such as a mainframe environment), or any source type that Federated Databases support.

Outline

1. Export the data from the source table into a file or a named pipe. Exporting to a named pipe provides the following benefits:
   - You can save space in the file system, since no intermediate data file is generated.
   - The exporting and loading/importing processes will be performed simultaneously.
2. If the source is not in the same database as the target, you have several choices for the database connection:
   - If the source database is on a distributed system, you can CATALOG it locally and then connect to it.
   - If the source database is on a system that is supported by DB2 Connect (for example, a mainframe system), you can CATALOG it locally and then connect to it (DB2 Connect will be used automatically). Appendix B, “DB2 connectivity” on page 307 contains examples on how to CATALOG a mainframe database.
   - If the source database in on a system that is supported by Federated databases, you can create a nickname for the source inside your target table’s database. Then you can connect to your target table’s database and export from the nickname. For details on setting up a Federated Database, see Appendix A, “Defining a Federated Database” on page 299.
3. There are restrictions on the file types allowed and whether or not using a named pipe is allowed. Restrictions are listed in the next section.
4. If possible, use the Load utility to move the data from the file or pipe into the target table, unless the amount of data to move is small (in which case, the import utility may perform faster). In some situations, it will be necessary to use the Import utility.

Evaluation

- Restrictions:
  - Only the IXF format is supported if the export is being performed through DB2 Connect. Additionally, if the target distributed system is in a partitioned database environment, you have the following choices:
    - If the environment is Version 8 or higher, load the IXF file using the LOAD_ONLY_VERIFY_PART mode (see “Loading IXF files in a Version 8 partitioned database environment” on page 97 for more details.)
    - Use the Import utility.
  - The Export and Load/Import must use the LOBSINFILE modifier if LOB objects greater than 32K in length are involved.
  - You cannot export into a named pipe in the following situations:
    - The LOBSINFILE modifier is being used.
    - A load is being performed in LOAD_ONLY_VERIFY_PART mode in a partitioned database environment.
  - For the Load or AutoLoader utility, see Chapter 6, “Load with DB2 Distributed” on page 91.

  - For the Export and Import utilities, see Chapter 5, “Export and Import with DB2 distributed” on page 79.
  - Formats to be used (considering both the restrictions just mentioned and those outlined in 11.1.1, “File format considerations” on page 269)
    - DEL
    - ASC
    - IXF
- Performance:
  - Using the fastest tool available on the target side
  - Data retrieved from the source through a select statement
- Usability:
  - The process is administered only on the distributed environment
  - Need to manually manage files or named pipes

Examples

The following examples demonstrate how the data can be moved from a mainframe system to distributed. However, as mentioned earlier, data can be moved from other source types, including *any* source that Federated Databases support.

Moving data from PAOLOR2.EMP to MARKOMP.EMP using a file

To move data from the host table PAOLOR2.EMP to the distributed table MARKOMP.EMP through exporting to a file, you can use the following CLP commands:

```
CONNECT TO hostdb USER paolor2 USING xxyyzz
EXPORT TO emp.ixf OF ixf MESSAGES export.msg select * from PAOLOR2.EMP
CONNECT RESET
CONNECT TO distdb USER markomp USING aabbcc
LOAD FROM emp.ixf OF ixf MESSAGES load.msg REPLACE INTO MARKOMP.EMP
```

In this example:

- Since a connection was made to a mainframe system, it was made through DB2 Connect. Therefore, the IXF file format was required.
- The Import utility could have been used instead of the Load utility. For small amounts of data, an import may be faster than a load due to the Load utility’s startup and shutdown overhead.

Moving data from PAOLOR2.EMP to MARKOMP.EMP using a named pipe

To move data from the host table PAOLOR2.EMP to the distributed table MARKOMP.EMP through exporting to a named pipe, you must first create the named pipe. For this example, we will assume that the distdb database (where the MARKOMP.EMP table belongs) resides on a UNIX distributed environment. We can create a named pipe called *mypipe* using the following system command:

```
mknod mypipe p
```

Next, to perform an export and a simultaneous load, you can run two separate CLP sessions simultaneously as follows:

- CLP Session 1:

```
CONNECT TO hostdb USER paolor2 USING xxyyzz
EXPORT TO mypipe OF ixf MESSAGES export.msg select * from PAOLOR2.EMP
CONNECT RESET
```

- CLP Session 2 (to import, replace “LOAD” with “IMPORT”):

```
CONNECT TO distdb USER markomp USING aabbcc
LOAD FROM mypipe OF ixf MESSAGES load.msg REPLACE INTO MARKOMP.EMP
CONNECT RESET
```

Note: Using a named pipe is only possible if the LOBSINFILE modifier is not used.

11.4 SQL insert containing a SELECT clause

This method of data movement is applicable to any source server type that Federated Databases support.

Outline

1. If the source does not exist in the same database as the target, use Federated Database support to create a nickname for the source (see Appendix A, “Defining a Federated Database” on page 299 for more details.)
2. It is recommended that the target table be created with the NOT LOGGED INITIALLY property specified, and that the ALTER TABLE command be executed against the table specifying APPEND ON. If there are any indexes defined on the table, it is recommended that they be dropped before the insert and recreated afterwards. Doing these things will help to optimize the performance of the insert.

Evaluation

- Restrictions:
  - DB2 UDB V8 has to be used on the distributed environment.
  - A Federated database must be used if the source is not in the same database as the target.
- Performance:
  - performance is likely suboptimal
- Usability:
  - The process is administered only on a distributed environment.
  - Since the source can be a nickname, the source can be anything that Federated databases support, even other database products. The only downside is that some additional setup is needed to create table nicknames when needed.
  - It is easy to use, and no intermediate data files are produced.
  - It has some benefits over using the Cross Loader, such as it can fire triggers where as the Load utility cannot.

Example

The following example demonstrates how an SQL insert can move data from a mainframe system to distributed. However, it can be used to move data from *any* source that Federated Databases support, to distributed.

Moving data from PAOLOR2.EMP to MARKOMP.EMP

To move data from the host table PAOLOR2.EMP to the distributed table MARKOMP.EMP (through the nickname MARKOMP.EMPHOST), you can issue the following SQL statement:

```
CONNECT TO distdb USER markomp USING aabbcc
INSERT INTO MARKOMP.EMP select * from MARKOMP.EMPHOST
```

In this example:

- In the Federated Database distdb, MARKOMP.EMPHOST was set up as a nickname for the table PAOLOR2.EMP that belonged to the mainframe database db2g (see 11.1.3, “Environment used for data movement examples” on page 271 for more details.)
- This example is applicable to both a partitioned database environment and a non-partitioned database environment.

11.5 Data Propagator

There are several redbook projects on Data Propagator (DPropR), the most current one is documented as a draft (Redpiece) in *The Practical Guide to DB2 Data Replication V8*, SG24-6828. We have not focused on DPropR in this project, but for readers with no experience, we want to highlight it together with the other functions and tools we recommend.The main reasons for using Data Propagator can be categorized in:

- Distribution of data to other locations
- Consolidation of data from other locations
- Bidirectional exchange of data with other locations

Replication from non-database sources is also provided through the federated data support.

Outline

Pre-propagation activities:

1. Define your tables as sources for the replication.
2. Create new tables as targets for the replication.
3. Populate control tables for Capture and Apply process.
4. Define subscription set and members.
5. Schedule Apply process.

Propagation activities (see Figure 11-4):

1. DB2 writes all changes in EMP table to the DB log
2. Capture program captures all the changes on the EMP table by reading the DB log
3. Captures writes committed changes to the EMPCD table and updates the REGISTER table
4. Apply program is invoked by addressing the actual SUBS-set. It reads new changes from the EMPCD table and inserts or updates the EMP target table. The following control tables will determine the processing of the changes:
   c. SUBS-member describes tables included in the subscription.
   d. SUBS-columns describes columns included in the tables.
   e. SUBS-statements describes before or after SQL to be executed during the apply phase.
5. Apply program updates SUBS_SET and PRUNCNTL tables.
6. Capture prunes delivered changes from the EMPCD table and updates PRUNCNTL table.

Because this is the most efficient, we have chosen to show the APPLY program using the PULL technique. APPLY could also run on the mainframe side and use the PUSH technique to insert the changes in the target table.

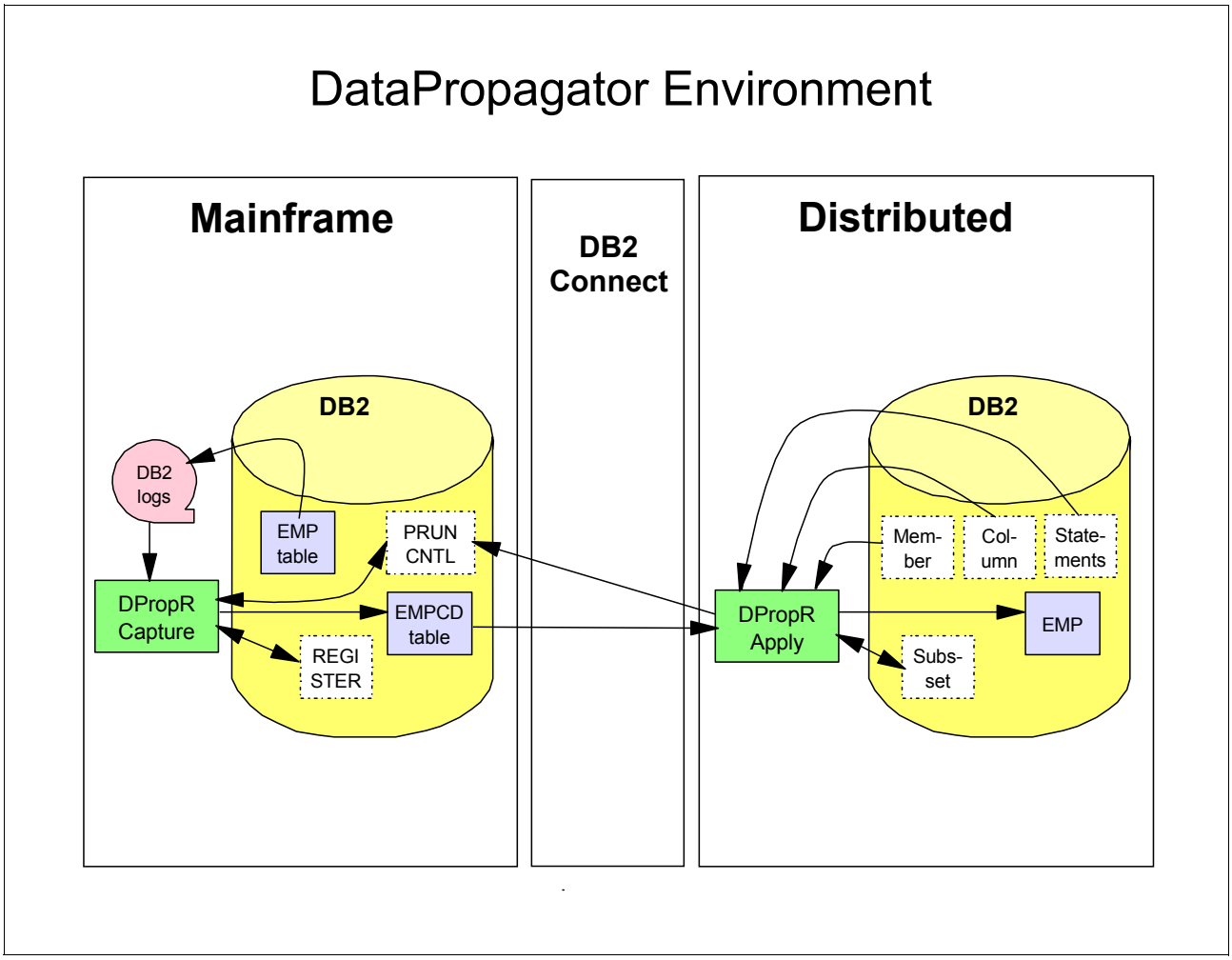

Figure 11-4 Data Propagator environment

Evaluation

Compared to other function and tools, DPropR is *not* appropriate for one time data movement. It is a tool that helps you keeping your databases in sync continuously or makes captured changes available. To facilitate this, a preparation and customizing effort is necessary.

You should consider DPropR if your need for data movement is based on continuously updated information like this:

- A complete or partial copy of a source table
- A complete or partial copy of changes to a source table
- An aggregate of a source table based on SQL column functions / GROUP BY
- An aggregate of changes to a source table based on SQL column functions / GROUP BY
- An updateable copy (replica) of all or portion of a source table

DPropR can also offer data transformation, filtering, complemental information, and derived information during the propagation process.

For other DBMS, the capture of changes is based on triggers fired inside time critical UOWs, but DPropR reads the undo and redo records in the database log. This does not influence the application.

Some of the enhancements in DB2 Replication V8

New features:

- Replication Center
- Replication Center Monitoring
- Replication Alert Monitor

Important enhancements:

- All stored passwords are encrypted.
- Changes are not inserted in CD-tables until they have been committed.
- No changes captured for your source unless the change affects your selected columns.
- Changes from the master site should not be recaptured at the replica site.
- Multiple Capture programs can run on the same DB2 database or subsystem.
- Capture prunes changes concurrently with the capture of changes on DB2.
- Faster full refreshes of target tables can be done using the load utility improvements.

A migration utility is included to convert existing DB2 Replication V5, V6, and V7 environments to DB2 Replication V8.

For a detailed discussion of enhancements, see *IBM DB2 Universal Database Replication Guide and Reference Version 8*, SC27-1121-00.

11.6 HPU for z/OS followed by Load or Import

This method of data movement is exclusive to using z/OS as the source.

Outline

1. HPU for z/OS can unload data into a delimited or a positional file. For the delimited file, make sure to specify that null indicators should not be enclosed by delimiters. For the positional file, make sure to specify external formats and padding.
   This are the only formats available on z/OS that can be read by utilities running on a distributed environment.
   You can unload either ASCII or EBCDIC.
2. Transfer the file from the host to the multiplatform workstation. If the unloaded files on the z/OS are in EBCDIC, specify ASCII as FTP option, else use BIN as option.
3. Use the Load utility to insert the data into the target table. If the target database is partitioned and you are using DB2 UDB V7 or older, use the AutoLoader to insert the data into the target table. Note that the Import utility can be used in this step. For small amounts of data, an import may be faster than a load due to the Load utility's startup and shutdown overhead.

Evaluation

- Restrictions:
  - For HPU, keep in mind that tables containing LOB objects are not supported. Also, see Chapter 7, "IBM DB2 High Performance Unload for z/OS" on page 119.
  - For the Load or AutoLoader utility, see Chapter 6, "Load with DB2 Distributed" on page 91.
  - For the Import utility, see Chapter 5, "Export and Import with DB2 distributed" on page 79.

  - Formats to be used:
    - Delimited format (DEL) is the easiest, but positional ASCII (ASC) created through the USER format is also an alternative. Consult 11.1.1, “File format considerations” on page 269 before making your choice.

- Performance:
  - These are the fastest tools available on both platforms (unless import is chosen).
- Usability:
  - Need to administer the process on both z/OS and multiplatform
  - Need to manually manage and transfer files

Examples

The following are examples on using the HPU for z/OS and then subsequently using the Load or Import utility to load the data into the distributed database *distdb*.

Unload EMP table in DELIMITED format with the HPU

Example 11-1 shows the JCL for unloading the table *PAOLOR2.EMP*. Following options are used:

DB2 NO	The job will terminate if HPU is not able to execute the SQL without using DB2
QUIESCE YES	All records will be written to disk before HPU accesses the VSAM data set
QUIESCECAT NO	Updates against the DB2 Catalog can still reside in the buffer pool, which probably means that the QUISCE information for EMP is not written to the VSAM data set with SYSIBM.SYSCOPY
DATE DATE_A	Columns with coltype will be given a predefined format (MM-DD-YYYY)
EBCDIC	The output data set will be EBCDIC encoded
DELIMITED	The format of the output data set will be DELIMITED
SEP ';'	**;** is chosen as the separator character
DELIM '*'	***** is chosen as the delimiter character
NULL DELIM	Defines that NULL values should not be enclosed by delimiters

Example 11-1 JCL HPU delimited format

```
//PAOLOHPU JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* HP UNLOAD EBCDIC - DELIMITED
//*****************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,DB2UNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//*
//SYSIN    DD  *
     UNLOAD TABLESPACE
      DB2 NO
      QUIESCE YES QUIESCECAT NO
      OPTIONS DATE DATE_A

       SELECT * FROM PAOLOR2.EMP
```

```
        OUTDDN (UNLDDN1) EBCDIC
        FORMAT DELIMITED SEP ';' DELIM '*' NULL DELIM
/*
//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//*
//********* DDNAMES USED BY THE SELECT STATEMENTS **********
//*
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.EBCDIDEL,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
```

In Example 11-2 the result from the HPU job is shown.

Example 11-2 HPU job report

```
******************************** TOP OF DATA ********************************
                 J E S 2  J O B  L O G  --  S Y S T E M  S C 6 3  --  N O D E

21.38.14 JOB11640 ---- TUESDAY,   05 NOV 2002 ----
21.38.14 JOB11640  IRR010I  USERID PAOLOR2  IS ASSIGNED TO THIS JOB.
21.38.14 JOB11640  ICH70001I PAOLOR2  LAST ACCESS AT 20:59:51 ON TUESDAY, NOVEMB
21.38.14 JOB11640  $HASP373 PAOLOHPU STARTED - INIT 1    - CLASS A - SYS SC63
21.38.14 JOB11640  IEF403I PAOLOHPU - STARTED - ASID=03EA - SC63
21.38.17 JOB11640  INZX006 EMPA1FHX TABLESPACE UNLOAD PHASE STARTED
21.38.17 JOB11640  INZX090 UNLDDN3  : 42 RECORDS WRITTEN
21.38.18 JOB11640  -                                        --TIMINGS (MINS.)--
21.38.18 JOB11640  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK
21.38.18 JOB11640  -PAOLOHPU          STEP1       00    585    .00    .00    .05
21.38.18 JOB11640  IEF404I PAOLOHPU - ENDED - ASID=03EA - SC63
21.38.18 JOB11640  -PAOLOHPU ENDED.  NAME-DB2UNLOAD            TOTAL CPU TIME=
.
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = DB2UNLOAD
DSNU050I    DSNUGUTC -  QUIESCE TABLESPACE DSNDB04.EMPA1FHX
DSNU477I  -DB2G DSNUQUIA - QUIESCE SUCCESSFUL FOR TABLESPACE DSNDB04.EMPA1FHX
DSNU474I  -DB2G DSNUQUIA - QUIESCE AT RBA 0002FAA28D32 AND AT LRSN 0002FAA28D32
DSNU475I    DSNUQUIB - QUIESCE UTILITY COMPLETE, ELAPSED TIME= 00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
INZX090 UNLDDN3  : 42 RECORDS WRITTEN
             TABLESPACE UNLOAD STATISTICS          UT4100 DB2 HIGH PERFORMANCE
* TABLE              * PART NO. * ROWS READ  * ROWS KEPT  * IX SCAN  *
*                    *          *            *            *          *
* EMP                *        0 *         42 *         42 *    0 %   *
* INVALID ROWS ..................*          0 *          0 *          *
* TABLESPACE TOTAL .............*         42 *         42 *    0 %   *
NUMBER OF PAGES READ ...      12
NUMBER OF PAGES IN ERROR       0
******************************** BOTTOM OF DATA ********************************
```

Example 11-3 shows how to FTP the file *PAOLOR2.MD.HPUNLOAD.EBCDIDEL* produced in Example 11-1. The file is EBCDIC encoded and you have to define ASCII conversion in the FTP. If the file is ASCII, you have to use the BIN option instead of ASCII.

Example 11-3 FTP the file of Example 11-1 from DOS window

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp 192.168.0.1<--- the IP-adress of mainframe
Connected to 192.168.0.1.
220-FTPMVS1 IBM FTP CS V1R2 at 192.168.0.1, 22:37:45 on 2002-11-05.
220 Connection will close if idle for more than 5 minutes.
User (192.168.0.1:(none)): paolor2
331 Send password please.
Password:       <--- type password
230 PAOLOR2 is logged on.  Working directory is "PAOLOR2.".
ftp> get 'paolor2.md.hpunload.ebcdidel' unload_ebcdic_del.txt ascii
200 Port request OK.
125 Sending data set PAOLOR2.MD.HPUNLOAD.EBCDIDEL
250 Transfer completed successfully.
```

Example 11-4 shows the file unload_ebcdic_del.txt after being FTP to the distributed system. Notice that NULL values are represented with **;;** and **;** if it is at the last column.

Example 11-4 The delimited file to be loaded on the distributed

```
*000010*;*CHRISTINE*;*I*;*HAAS*;*A00*;;1965-01-01;*PRES    *; 18;*F*;1933-08-14; 52750.00; 1000.00; 4220.00
*000020*;*MICHAEL*;*L*;*THOMPSON*;*B01*;*3476*;1973-10-10;*MANAGER *; 18;;1948-02-02; 41250.00;; 3300.00
*000030*;*SALLY*;*A*;*KWAN*;*C01*;*4738*;1975-04-05;*MANAGER *; 20;*F*;1941-05-11; 38250.00; 800.00;
*000050*;*JOHN*;*B*;*GEYER*;*E01*;*6789*;1949-08-17;; 16;*M*;1925-09-15; 40175.00; 800.00; 3214.00
*000060*;*IRVING*;*F*;*STERN*;*D11*;*6423*;1973-09-14;*MANAGER *; 16;*M*;1945-07-07; 32250.00; 600.00; 2580.00
*000070*;*EVA*;*D*;*PULASKI*;*D21*;*7831*;1980-09-30;; 16;*F*;1953-05-26; 36170.00; 700.00; 2893.00
*000090*;*EILEEN*;*W*;*HENDERSON*;*E11*;*5498*;1970-08-15;; 16;*F*;1941-05-15; 29750.00; 600.00; 2380.00
*000100*;*THEODORE*;*Q*;*SPENSER*;*E21*;*0972*;1980-06-19;; 14;*M*;1956-12-18; 26150.00; 500.00; 2092.00
*000110*;*VINCENZO*;*G*;*LUCCHESI*;*A00*;*3490*;1958-05-16;; 19;*M*;1929-11-05; 46500.00;-900.00; 3720.00
*000120*;*SEAN*;* *;*O'CONNELL*;*A00*;*2167*;1963-12-05;*CLERK   *; 14;*M*;1942-10-18; 29250.00; 600.00; 2340.00
```

Load or import unload_ebcdic_del.txt into distributed

Loading unload_ebcdic_del.txt into the MARKOMP.EMP table can be accomplished using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
load from unload_ebcdic_del.txt of del modified by chardel0x2A coldel0x3B replace into
MARKOMP.EMP
```

Note: The character delimiter 0x2A (asterisk) and column delimiter 0x3B (semicolon) are specified in the load command.

For Version 7, if you are using the db2atld utility in a partitioned database environment, the above Load command can be used in the autoloader configuration file.

Importing into MARKOMP.EMP can be accomplished by using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
import from unload_ebcdic_del.txt of del modified by chardel0x2A coldel0x3B replace into
MARKOMP.EMP
```

Unload EMP table in POSITIONAL format with HPU

Example 11-5 shows the JCL for unloading the table PAOLOR2.EMP into a positional file. The following options are used:

DB2 NO	The job will terminate if HPU is not able to execute the SQL without using DB2.
QUIESCE YES	All records will be written to disk before HPU accesses the VSAM data set.
QUIESCECAT NO	Updates against the DB2 Catalog can still reside in the buffer pool, which probably means that the QUIESCE information for EMP is not written to the VSAM data set with SYSIBM.SYSCOPY.
NULL X'FF' X'00'	A NULL indicator is included in the output data set for all columns that can contain NULLS. If NULL a hex high value (x'FF') will be generated, if not NULL a hex low value (x'00') will be generated.
PIC ('+', LEAD, '.')	Defines the external display format of numeric data. '+' means that positive values are given a '+' and negative values are given a '-'. LEAD places the sign in front of the numeric value. '.' is the decimal separator.
DATE DATE_A	Columns with coltype will be given a predefined format (MM-DD-YYYY)
EBCDIC	The output data set will be EBCDIC encoded.

Columns that need to be treated different than the default, must all be specified in the user block:

TYPE	In this example, the VARCHAR columns are converted to CHAR and given the maximum length. The SMALINT and DECIMAL columns are converted to CHAR and given a length that includes sign and decimal separator.
PADDING	If the contents of the column is less than the maximum, ' ' (blanks) are padded to the left (see next).
JUST LEFT	Specifies that the output string is justified to the left

Example 11-5 JCL HPU positional format

```
//PAOLOHPU JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//* HP UNLOAD EBCDIC - USERDEFINED (FIXED)
//*************************************************************
//*
//STEP1    EXEC PGM=INZUTILB,REGION=0M,DYNAMNBR=99,
//         PARM='DB2G,DB2UNLOAD'
//STEPLIB  DD  DSN=INZ.V2R1M0.SINZLINK,DISP=SHR
//         DD  DSN=DB2V710G.SDSNEXIT,DISP=SHR
//         DD  DSN=DB2G7.SDSNLOAD,DISP=SHR
//*
//SYSIN    DD  *
     UNLOAD TABLESPACE
      DB2 NO
      QUIESCE YES QUIESCECAT NO
      OPTIONS NULL X'FF' X'00' PIC ('+', LEAD, '.') DATE DATE_A

     SELECT * FROM PAOLOR2.EMP
     ORDER BY 1
       OUTDDN (UNLDDN1) EBCDIC
       FORMAT USER
         (
         COL FIRSTNME  TYPE CHAR(12) PADDING ' ' JUST LEFT,
         COL LASTNAME  TYPE CHAR(15) PADDING ' ' JUST LEFT,
         COL EDLEVEL   TYPE CHAR(6),
         COL SALARY    TYPE CHAR(13),
```

```
         COL BONUS     TYPE CHAR(13),
         COL COMM      TYPE CHAR(13)
         )
       LOADDDN LOADDDN1

/*
//SYSPRINT DD  SYSOUT=*
//COPYDD   DD  SYSOUT=*
//OUTDD    DD  SYSOUT=*
//UTPRINT  DD  SYSOUT=*
//LOADDDN1 DD  SYSOUT=*
//*
//********* DDNAMES USED BY THE SELECT STATEMENTS **********
//*
//UNLDDN1  DD  DSN=PAOLOR2.MD.HPUNLOAD.EBCDIFIX,
//             DISP=(NEW,CATLG,CATLG),
//             UNIT=SYSDA,
//             SPACE=(TRK,(10,20),RLSE)
```

Example 11-6 shows the load statements generated. This will be useful when defining the column list pairs to the Load utility on the distributed.

Example 11-6 Load statements positional format generated of HPU

```
LOAD DATA LOG NO  ENFORCE NO EBCDIC CCSID(007)
INTO TABLE PAOLOR2.EMP
(
EMPNO POSITION (     1  )         CHAR (     6 ),
FIRSTNME POSITION (     7  )         CHAR (    12 ),
MIDINIT POSITION (    19  )         CHAR (     1 ),
LASTNAME POSITION (    20  )         CHAR (    15 ),
WORKDEPT POSITION (    36  )         CHAR (     3 )
     NULLIF(    35 ) = X'FF',
PHONENO POSITION (    40  )         CHAR (     4 )
     NULLIF(    39 ) = X'FF',
HIREDATE POSITION (    45  )         DATE EXTERNAL (    10 )
     NULLIF(    44 ) = X'FF',
JOB POSITION (    56  )         CHAR (     8 )
     NULLIF(    55 ) = X'FF',
EDLEVEL POSITION (    65  )         INTEGER EXTERNAL (     6 )
     NULLIF(    64 ) = X'FF',
SEX POSITION (    72  )         CHAR (     1 )
     NULLIF(    71 ) = X'FF',
BIRTHDATE POSITION (    74  )         DATE EXTERNAL (    10 )
     NULLIF(    73 ) = X'FF',
SALARY POSITION (    85  )         DECIMAL EXTERNAL (    13,  2 )
     NULLIF(    84 ) = X'FF',
BONUS POSITION (    99  )         DECIMAL EXTERNAL (    13,  2 )
     NULLIF(    98 ) = X'FF',
COMM POSITION (   113  )         DECIMAL EXTERNAL (    13,  2 )
     NULLIF(   112 ) = X'FF',
)
```

Example 11-7 is an example how to FTP the file *PAOLOR2.MD.HPUNLOAD.EBCDIFIX* produced in Example 11-5. The file is EBCDIC encoded and you have to define ASCII conversion in the FTP. If the file is ASCII, you have to use the BIN option instead of ASCII.

Example 11-7 FTP from a DOS window

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp 192.168.0.1<--- the IP-adress of mainframe
Connected to 192.168.0.1.
220-FTPMVS1 IBM FTP CS V1R2 at 192.168.0.1, 22:37:45 on 2002-11-05.
220 Connection will close if idle for more than 5 minutes.
User (192.168.0.1:(none)): paolor2
331 Send password please.
Password:       <--- type password
230 PAOLOR2 is logged on.  Working directory is "PAOLOR2.".
ftp> get 'paolor2.md.hpunload.ebcdifix' unload_ebcdic_fixed.hpu.txt ascii
200 Port request OK.
125 Sending data set PAOLOR2.MD.HPUNLOAD.EBCDIFIX
250 Transfer completed successfully.
```

Example 11-8 shows the file unload_ebcdic_fixed.hpu.txt to be loaded on the distributed side after being converted to ASCII during the FTP. The NULL indicators appears as a 'ÿ'.

Example 11-8 Positioned output file from HPU

```
000010CHRISTINE   IHAAS           A00ÿ     01-01-1965 PRES     +00018 F 08-14-1933 +000052750.00 +000001000.00 +000004220.00
000020MICHAEL     LTHOMPSON       B01 3476 10-10-1973 MANAGER  +00018ÿ  02-02-1948 +000041250.00ÿ              +000003300.00
000030SALLY       AKWAN           C01 4738 04-05-1975 MANAGER  +00020 F 05-11-1941 +000038250.00 +000000800.00ÿ
000050JOHN        BGEYER          E01 6789 08-17-1949ÿ         +00016 M 09-15-1925 +000040175.00 +000000800.00 +000003214.00
000060IRVING      FSTERN          D11 6423 09-14-1973 MANAGER  +00016 M 07-07-1945 +000032250.00 +000000600.00 +000002580.00
000070EVA         DPULASKI        D21 7831 09-30-1980ÿ         +00016 F 05-26-1953 +000036170.00 +000000700.00 +000002893.00
000090EILEEN      WHENDERSON      E11 5498 08-15-1970ÿ         +00016 F 05-15-1941 +000029750.00 +000000600.00 +000002380.00
000100THEODORE    QSPENSER        E21 0972 06-19-1980ÿ         +00014 M 12-18-1956 +000026150.00 +000000500.00 +000002092.00
000110VINCENZO    GLUCCHESI       A00 3490 05-16-1958ÿ         +00019 M 11-05-1929 +000046500.00 -000000900.00 +000003720.00
000120SEAN         O'CONNELL      A00 2167 12-05-1963 CLERK    +00014 M 10-18-1942 +000029250.00 +000000600.00 +000002340.00
```

Load or import unload_ebcdic_fixed.hpu.txt into distributed

Loading unload_ebcdic_fixed.hpu.txt into the MARKOMP.EMP table can be accomplished using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
load from unload_ebcdic_fixed.hpu.txt of ASC modified by nullindchar=0xFF STRIPTBLANKS
method l (1 6, 7 18, 19 19, 20 34, 36 38, 40 43, 45 54, 56 63, 65 70, 72 72, 74 83, 85
97, 99 111, 113 125) null indicators (0, 0, 0, 0, 35, 39, 44, 55, 64, 71, 73, 84, 98,
112) replace into MARKOMP.EMP
```

Note: The flexible length positional ASC format is used here. This is indicated by the lack of the RECLEN modifier being used in the load command. Also notice that the STRIPTBLANKS modifier is being used here. Without specifying it, VARCHAR column data would have been padded with trailing blanks.

For Version 7, if you are using the db2atld utility in a partitioned database environment, the above load command can be used in the autoloader configuration file.

Importing into MARKOMP.EMP can be accomplished by using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
import from unload_ebcdic_fixed.hpu.txt of ASC modified by nullindchar=0xFF STRIPTBLANKS
method l (1 6, 7 18, 19 19, 20 34, 36 38, 40 43, 45 54, 56 63, 65 70, 72 72, 74 83, 85
97, 99 111, 113 125) null indicators (0, 0, 0, 0, 35, 39, 44, 55, 64, 71, 73, 84, 98,
112) replace into MARKOMP.EMP
```

11.7 Unload utility followed by Load or Import

This method of data movement is exclusive to using z/OS as the source.

Outline

1. Unload utility on z/OS can unload data into a positional file. Make sure to specify external formats and keep the default padding.
   You can unload either ASCII or EBCDIC.
2. Transfer the file from the host to the distributed system. If the unloaded files on the z/OS are in EBCDIC, specify ASCII as FTP option, else use BIN as option.
3. Use the Load utility to insert the data into the target table. If the target database is partitioned and you are using DB2 UDB V7 or older, use the AutoLoader to insert the data into the target table. Note that the import utility can be used in this step. For small amounts of data, an import may be faster than a load due to the Load utility's startup and shutdown overhead.

Evaluation

- Restrictions:
  - For Unload utility, see Chapter 3, "Unload with DB2 for z/OS" on page 33.
  - For the Load or AutoLoader utility, see Chapter 6, "Load with DB2 Distributed" on page 91.
  - For the Import utility, see Chapter 5, "Export and Import with DB2 distributed" on page 79.
  - Formats to be used
    - Positional ASCII (ASC). Be aware of the restrictions outlined in 11.1.1, "File format considerations" on page 269.
- Performance:
  - Using fastest tools available on both platforms (unless import is chosen)
- Usability:
  - Need to administer the process on both z/OS and multiplatform
  - Need to manually manage and transfer files

Examples

The following are examples on using the Unload tool, and then subsequently using the Load or Import utility to load the data into the distributed database *distdb*.

Unload EMP table in POSITIONAL format with the Unload utility

Example 11-9 shows the JCL for the unload utility to run. You have to explicit define EXTERNAL formats on data types that needs to be converted before the loaded. Be sure to use NOSUBS if you unload with ASCII option. Padding is default, so never use NOPAD option. The SYSPUNCH makes the load statement available.

Example 11-9 JCL Unload utility positional format

```
//PAOLUNLO JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*************************************************************
//* UNLOAD EBCDIC PADDED "EXTERNAL FORMAT" ON SOME COLUMNS
//*************************************************************
```

```
//*
//STEP1    EXEC DSNUPROC,UID='UNLOAD',UTPROC='',SYSTEM='DB2G'
//SYSREC   DD DSN=PAOLOR2.MD.UNLOAD.EBCDIFIX,
//         DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(2,1),RLSE)
//SYSPUNCH DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  UNLOAD DATA NOSUBS
  FROM TABLE PAOLOR2.EMP
  (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE,
  JOB, EDLEVEL INTEGER EXTERNAL, SEX, BIRTHDATE, SALARY DECIMAL
  EXTERNAL, BONUS DECIMAL EXTERNAL, COMM DECIMAL EXTERNAL)
  PUNCHDDN SYSPUNCH UNLDDN SYSREC EBCDIC
//*
```

Example 11-10 shows the result from the Unload utility run. The load statements with positions can be useful when defining the input file for the Load utility.

Example 11-10 Unload utility job report

```
******************************** TOP OF DATA ********************************
                 J E S 2  J O B  L O G  --  S Y S T E M  S C 6 3  --  N O D E

13.28.55 JOB11693 ---- WEDNESDAY, 06 NOV 2002 ----
13.28.55 JOB11693  IRR010I  USERID PAOLOR2  IS ASSIGNED TO THIS JOB.
13.28.55 JOB11693  ICH70001I PAOLOR2  LAST ACCESS AT 13:26:38 ON WEDNESDAY, NOVE
13.28.55 JOB11693  $HASP373 PAOLUNLO STARTED - INIT 1    - CLASS A - SYS SC63
13.28.55 JOB11693  IEF403I PAOLUNLO - STARTED - ASID=03EA - SC63
13.28.59 JOB11693  -                                          --TIMINGS (MINS.)--
13.28.59 JOB11693  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  CLOCK
13.28.59 JOB11693  -PAOLUNLO STEP1    DSNUPROC    00    248    .00    .00    .06
13.28.59 JOB11693  IEF404I PAOLUNLO - ENDED - ASID=03EA - SC63
13.28.59 JOB11693  -PAOLUNLO ENDED.  NAME-DB2UNLOAD            TOTAL CPU TIME=
13.28.59 JOB11693  $HASP395 PAOLUNLO ENDED
.
DSNU253I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=42 FOR TABLE
PAOLOR2.EMP
DSNU252I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=42 FOR
TABLESPACE DSNDB04.EMPA1FHX
DSNU250I    DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU010I    DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0

LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "PAOLOR2 "."EMP              "
 WHEN(00001:00002 = X'01B7')

 ( "EMPNO            " POSITION(  00003:00008) CHAR(006)
 , "FIRSTNME         " POSITION(  00009:00022) VARCHAR
 , "MIDINIT          " POSITION(  00023:00023) CHAR(001)
 , "LASTNAME         " POSITION(  00024:00040) VARCHAR
 , "WORKDEPT         " POSITION(  00042:00044) CHAR(003)
                         NULLIF(00041)=X'FF'
 , "PHONENO          " POSITION(  00046:00049) CHAR(004)
                         NULLIF(00045)=X'FF'
 , "HIREDATE         " POSITION(  00051:00060) DATE EXTERNAL
                         NULLIF(00050)=X'FF'
 , "JOB              " POSITION(  00062:00069) CHAR(008)
```

```
                          NULLIF(00061)=X'FF'
 , "EDLEVEL           " POSITION(  00071:00081) INTEGER EXTERNAL(011)
                          NULLIF(00070)=X'FF'
 , "SEX               " POSITION(  00083:00083) CHAR(001)
                          NULLIF(00082)=X'FF'
 , "BIRTHDATE         " POSITION(  00085:00094) DATE EXTERNAL
                          NULLIF(00084)=X'FF'
 , "SALARY            " POSITION(  00096:00106) DECIMAL EXTERNAL(11,02)
                          NULLIF(00095)=X'FF'
 , "BONUS             " POSITION(  00108:00118) DECIMAL EXTERNAL(11,02)
                          NULLIF(00107)=X'FF'
 , "COMM              " POSITION(  00120:00130) DECIMAL EXTERNAL(11,02)
                          NULLIF(00119)=X'FF'
 )
******************************** BOTTOM OF DATA ********************************
```

Attention: To correct position error in the generated load statements (only when including varchar columns), make sure that PTF XXXX for APAR PQ66712 is applied to your system.

Example 11-11 is an example how to FTP the file *PAOLOR2.MD.UNLOAD.EBCDIFIX* produced in Example 11-9. The file is EBCDIC encoded and you have to define ASCII conversion in the FTP. If the file is ASCII, you have to use the BIN option instead of ASCII.

Example 11-11 FTP the file of Example 11-9 from a DOS window

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp 192.168.0.1<--- the IP-adress of mainframe
Connected to 192.168.0.1.
220-FTPMVS1 IBM FTP CS V1R2 at 192.168.0.1, 22:37:45 on 2002-11-05.
220 Connection will close if idle for more than 5 minutes.
User (192.168.0.1:(none)): paolor2
331 Send password please.
Password:       <--- type password
230 PAOLOR2 is logged on.  Working directory is "PAOLOR2.".
ftp> get 'paolor2.md.unload.ebcdifix' unload_ebcdic_fixed.txt ascii
200 Port request OK.
125 Sending data set PAOLOR2.MD.UNLOAD.EBCDIFIX
250 Transfer completed successfully.
```

Example 11-12 shows the output file unload_ebcdic_fixed.txt from the Unload after it is transferred and converted to ASCII on the distributed.

Example 11-12 The positional file to be loaded on the distributed

```
» 000010 CHRISTINE   I HAAS           A00ÿ     1965-01-01 PRES     18          F 1933-08-14 52750.00    1000.00     4220.00
» 000020 MICHAEL     L THOMPSON       B01 3476 1973-10-10 MANAGER  18         ÿ  1948-02-02 41250.00    ÿ           3300.00
» 000030 SALLY       A KWAN           C01 4738 1975-04-05 MANAGER  20          F 1941-05-11 38250.00    800.00      ÿ
» 000050 JOHN        B GEYER          E01 6789 1949-08-17ÿ         16          M 1925-09-15 40175.00    800.00      3214.00
» 000060 IRVING      F STERN          D11 6423 1973-09-14 MANAGER  16          M 1945-07-07 32250.00    600.00      2580.00
» 000070 EVA         D PULASKI        D21 7831 1980-09-30ÿ         16          F 1953-05-26 36170.00    700.00      2893.00
» 000090 EILEEN      W HENDERSON      E11 5498 1970-08-15ÿ         16          F 1941-05-15 29750.00    600.00      2380.00
» 000100 THEODORE    Q SPENSER        E21 0972 1980-06-19ÿ         14          M 1956-12-18 26150.00    500.00      2092.00
» 000110 VINCENZO    G LUCCHESI       A00 3490 1958-05-16ÿ         19          M 1929-11-05 46500.00    -900.00     3720.00
» 000120 SEAN          O'CONNELL      A00 2167 1963-12-05 CLERK    14          M 1942-10-18 29250.00    600.00      2340.00
```

Load or import unload_ebcdic_fixed.txt into distributed

Loading unload_ebcdic_fixed.txt into the MARKOMP.EMP table can be accomplished using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
load from unload_ebcdic_fixed.txt of ASC modified by nullindchar=0xFF reclen=131
STRIPTBLANKS method l (3 8, 11 22, 23 23, 26 40, 42 44, 46 49, 51 60, 62 69, 71 81, 83
83, 85 94, 96 106, 108 118, 120 130) null indicators (0, 0, 0, 0, 41, 45, 50, 61, 70,
82, 84, 95, 107, 119) replace into MARKMO.EMP
```

Note: The fixed length positional ASC format is used here. This is indicated by the use of the RECLEN modifier in the **load** command. Also notice that the STRIPTBLANKS modifier is being used here. Without specifying it, VARCHAR column data would have been padded with trailing blanks.

For Version 7, if you are using the db2atld utility in a partitioned database environment, the above **load** command can be used in the autoloader configuration file.

Importing into MARKOMP.EMP can be accomplished by using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
import from unload_ebcdic_fixed.txt of ASC modified by nullindchar=0xFF reclen=131
STRIPTBLANKS method l (3 8, 11 22, 23 23, 26 40, 42 44, 46 49, 51 60, 62 69, 71 81, 83
83, 85 94, 96 106, 108 118, 120 130) null indicators (0, 0, 0, 0, 41, 45, 50, 61, 70,
82, 84, 95, 107, 119) replace into MARKMO.EMP
```

Unload LITERATURE table (with LOB data) in POSITIONAL format with the unload utility

Example 11-13 shows the JCL for the Unload utility to run. LOB data columns must be less than 32 KB in length, since this is the limit for inline data used with Import and Load. This limit can be overcome by using the LOBSINFILE option.

Example 11-13 The Unload utility JCL

```
//PAOLUNLO JOB  (999,POK),'DB2UNLOAD',CLASS=A,MSGCLASS=T,
//    NOTIFY=&SYSUID,TIME=1440,REGION=0M,MSGLEVEL=(1,1)
//*
//*****************************************************************
//* UNLOAD EBCDIC LOB
//*****************************************************************
//*
//STEP1    EXEC DSNUPROC,UID='UNLOAD',UTPROC='',SYSTEM='DB2G'
//SYSREC   DD DSN=PAOLOR2.MD.UNLOAD.EBCDILOB,
//         DISP=(OLD,CATLG,CATLG),
//         UNIT=SYSDA,SPACE=(TRK,(2,1),RLSE)
//SYSPUNCH DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  UNLOAD DATA
  FROM TABLE SC24630O.LITERATURE
  PUNCHDDN SYSPUNCH UNLDDN SYSREC EBCDIC
//*
```

Example 11-14 shows the load statements for the LITERATURE table.

Example 11-14 LOAD statement from PUNCHDDN

```
LOAD DATA INDDN SYSREC   LOG NO  RESUME YES
 EBCDIC  CCSID(00037,00000,00000)
 INTO TABLE "SC246300"."LITERATURE        "
 WHEN(00001:00002 = X'006F')
 ( "TITLE             " POSITION(  00004:00028) CHAR(025)
                          NULLIF(00003)=X'FF'
 , "MOVLENGTH         " POSITION(  00072:00075) INTEGER
                          NULLIF(00071)=X'FF'
 , "LOBMOVIE          " POSITION(  00077:02128) BLOB
                          NULLIF(00076)=X'FF'
 , "LOBBOOK           " POSITION(  02130:12373) CLOB
                          NULLIF(02129)=X'FF'
 )
```

Example 11-15 is an example how to FTP the file *PAOLOR2.MD.UNLOAD.EBCDILOB* produced in Example 11-13. The file is EBCDIC encoded and you have to define ASCII conversion in the FTP. If the file is ASCII, you have to use the BIN option instead of ASCII.

Example 11-15 FTP file of Example 11-13 from a DOS window

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>ftp 192.168.0.1<--- the IP-adress of mainframe
Connected to 192.168.0.1.
220-FTPMVS1 IBM FTP CS V1R2 at 192.168.0.1, 22:37:45 on 2002-11-05.
220 Connection will close if idle for more than 5 minutes.
User (192.168.0.1:(none)): paolor2
331 Send password please.
Password:       <--- type password
230 PAOLOR2 is logged on.  Working directory is "PAOLOR2.".
ftp> get 'paolor2.md.unload.ebcdilob' unload_ebcdic_fixed.lob.txt ascii
200 Port request OK.
125 Sending data set PAOLOR2.MD.UNLOAD.EBCDILOB
250 Transfer completed successfully.
```

Load or import unload_ebcdic_fixed.lob.txt into distributed

Loading unload_ebcdic_fixed.lob.txt into the MARKOMP.LIT table can be accomplished using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
load from unload_ebcdic_fixed.lob.txt of ASC modified by nullindchar=0xFF STRIPTBLANKS
method l (4 28, 30 30, 72 75, 77 2128, 2134 12373) null indicators (3, 0, 71, 76, 2129)
replace into MARKOMP.LIT
```

Note: The flexible length positional ASC format is used here. This is indicated by the lack of the RECLEN modifier being used in the load command. Also notice that the STRIPTBLANKS modifier is being used here. Without specifying it, VARCHAR column data would have been padded with trailing blanks.

For Version 7, if you are using the db2atld utility in a partitioned database environment, the above **load** command can be used in the autoloader configuration file.

Importing into MARKOMP.LIT can be accomplished by using the following CLP commands:

```
CONNECT TO distdb USER markomp USING aabbcc
import from unload_ebcdic_fixed.lob.txt of ASC modified by nullindchar=0xFF STRIPTBLANKS
method l (4 28, 30 30, 72 75, 77 2128, 2134 12373) null indicators (3, 0, 71, 76, 2129)
replace into MARKOMP.LIT
```

11.8 HPU for MP followed by Import or Load

This method of data movement is applicable when moving data within a distributed environment.

Outline

1. Use the HPU to unload the data from the source table into a file or a named pipe. Unloading to a named pipe provides the following benefits:
   - You can save space in the file system, since no intermediate data file is generated.
   - The exporting and loading/importing processes will be performed simultaneously.
2. The source table has to reside on the same machine where HPU runs.
3. There are restrictions on the file types allowed and whether or not using a named pipe is allowed. Restrictions are listed in the next section.
4. If possible, use the Load utility to move the data from the file or pipe into the target table, unless the amount of data to move is small (in which case, the import utility may perform faster). In some situations, it will be necessary to use the Import utility.

Evaluation

- Restrictions:
  - For the HPU for MP, see Chapter 8, “IBM DB2 High Performance Unload for Multiplatforms” on page 193.
  - For the Load or AutoLoader utility, see Chapter 6, “Load with DB2 Distributed” on page 91.
  - For the Import utility, see Chapter 5, “Export and Import with DB2 distributed” on page 79.
  - Do not specify a named pipe to unload into if you are generating LOB files.
  - Currently supported formats are (see 11.1.1, “File format considerations” on page 269 before choosing which file format to use):
    - DEL
    - IXF

    Note: For the SELECT statement used, it is recommended that you use one that is a basic “SELECT * from table” or “SELECT col1,col2,etc from table”. If you specify a statement that is too complex, HPU will invoke the Export utility, which in most cases is not as fast.

- Performance:
  - Using the fastest tool available on both, source and target side
  - Data retrieved from the source through a select statement only if
- Usability:
  - The process is administered only on the distributed environment
  - Need to manually manage files or named pipes

Examples

The following examples demonstrate how the data can be moved between distributed systems.

Moving data from MIKEMP.EMP to MARKOMP.EMP using a file

To move data from the table MIKEMP.EMP to the table MARKOMP.EMP through HP unload to a file, you can chose one of the following three examples:

- In example 1 we are working from two machines.

  First, on the machine where distdb2 resides, run the following command to unload the contents of the MIKEMP.EMP table into a data file emp.del:

  ```
  db2hpu -d distdb2 -m hpunload.txt -o emp.del -t mikemp.emp -i db2inst3
  ```

  Next, move the unloaded data file emp.del to the machine where distdb resides through one of the following methods:

  – FTPing the emp.del file to the machine.
  – If you are in Windows, map the drive where emp.del was unloaded.

  Next, load the data file using the following CLP commands (on the machine where distdb resides):

  ```
  CONNECT TO distdb USER markomp USING aabbcc
  LOAD FROM emp.del OF del MESSAGES import.msg REPLACE INTO MARKOMP.EMP
  ```

- In example 2 we are working from the machine where distdb2 resides.

  First, run the following command to unload the contents of the MIKEMP.EMP table into a data file emp.del:

  ```
  db2hpu -d distdb2 -m hpunload.txt -o emp.del -t mikemp.emp -i db2inst3
  ```

  Next, import the data file using the following CLP commands:

  ```
  CONNECT TO distdb USER markomp USING aabbcc
  IMPORT FROM emp.del OF del MESSAGES import.msg REPLACE INTO MARKOMP.EMP
  ```

- In example 3 we are also working from the machine where distdb2 resides.

  First, run the following command to unload the contents of the MIKEMP.EMP table into a data file emp.del:

  ```
  db2hpu -d distdb2 -m hpunload.txt -o emp.del -t mikemp.emp -i db2inst3
  ```

  Next, load the data file using the following CLP commands:

  ```
  CONNECT TO distdb USER markomp USING aabbcc
  LOAD CLIENT FROM emp.del OF del MESSAGES import.msg REPLACE INTO MARKOMP.EMP
  ```

  In example 3 we are using the CLIENT option, which allows us to instruct the load utility to load data that exists on our client (the machine where distdb2 resides). Execute the load command from the machine where distdb2 resides, while you are connected to a locally cataloged version of the distdb database. emp.del exists on the machine where distdb2 resides.

Moving data from MIKEMP.EMP to MARKOMP.EMP using a named pipe

To move data from the host table MIKEMP.EMP to the distributed table MARKOMP.EMP through HP unload to a named pipe, you must first create the named pipe. For this example, we will assume that the distdb database (where the tables belongs) resides on a UNIX distributed environment. We can create a named pipe called *mypipe* using the following system command:

```
mknod mypipe p
```

Next, to perform an HP unload and a simultaneous load, you can run two separate sessions simultaneously as follows:

- Session 1 (system command):

```
db2hpu -d distdb2 -m hpunload.txt -o mypipe -t mikemp.emp -i db2inst3
```

- Session 2 (CLP):

```
CONNECT TO distdb USER markomp USING aabbcc
IMPORT FROM mypipe OF ixf MESSAGES load.msg REPLACE INTO MARKOMP.EMP
CONNECT RESET
```

Part 5

Appendixes

A

Defining a Federated Database

Federated Databases in a distributed environment provide a very effective means of moving data. They are available on DB2 UDB Version 7 and above:

- They can be used in conjunction with the Export utility or the Cross Loader (on a distributed platform) to move data out of any system that Federated Databases support.
- Data movement can also be performed through Federated Databases by executing an SQL insert statement that contains a SELECT clause.
- It is even possible to perform a cross load on a mainframe environment that reads data from a Federated Database through a three part name.

Version 8 includes many enhancements, the most important are:

- Additional data sources and operating systems

  You can access Informix data sources, and, through DB2 Relational Connect, you can access Microsoft SQL Server, Sybase, and ODBC data sources. The new Life Sciences Data Connect enables you to access BLAST search algorithms, Documentum data files, Microsoft Excel spreadsheets, table-structured files, and XML tagged files. All DB2 servers using Linux, HP-UX, and Windows 2000 operating systems can now be federated servers.

- Write capability (or transparent DDL)

  You can now issue INSERT, UPDATE, and DELETE statements on nicknames. This enables DB2 replication to support replication to and from non-DB2 data sources. You can also create remote tables on relational data sources.

- New federated DB2 Control Center support

  Using the DB2 Control Center you can quickly create the wrappers, supply the server definitions, identify user mappings, and create nicknames for the data source objects. Additionally, the DB2 Control Center is the easiest way to create remote tables using transparent DDL.

For more details on federated systems and Federated Databases, and of course specifics on the new V8 functions, refer to the *DB2 UDB Federated Systems Guide Version 8,* SC09-4830.

A.1 Examples of creating Federated Databases

A Federated Database can be created using the following steps:

1. Update DATABASE MANAGER CONFIGURATION using FEDERATED YES.
2. CATALOG a node against the remote system.
3. Locally CATALOG the remote database against the node (not necessary for Version 7).
4. Create a database that will be used as the Federated Database, and CONNECT to it.
5. Create a WRAPPER that corresponds with the remote system.
6. Create a SERVER against the remote system that refers to the node just created, and the database containing the remote system. Note that Version 7 and Version 8 differ in the way that the database is referenced.
7. Create a USER MAPPING for the instance user against the SERVER just created.
8. Create a NICKNAME for the remote source.

What follows are examples on how to set up Federated Databases in both v7 and v8 using CLP commands, against both a distributed environment and a mainframe environment, see Figure A-1.

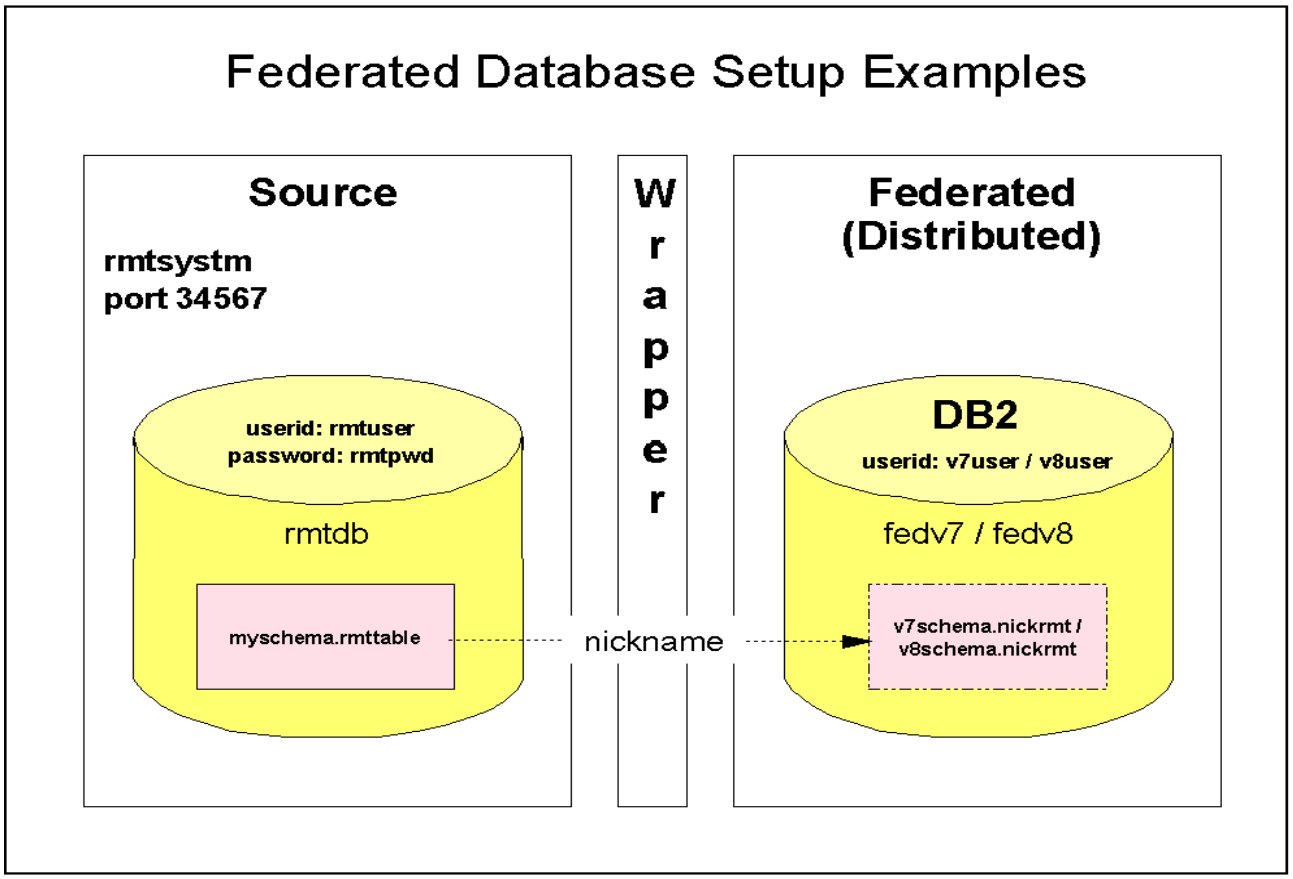

Figure A-1 Federated database setup examples

A.1.1 Federated database setup with DB2 V7

We show the setup steps for accessing both, the distributed and the mainframe environments.

A.1.1.1 Against a distributed environment

Suppose an instance user **v7user** wishes to set up a Federated Database **fedv7** to be able to access a table **myschema.rmttable** through a *nickname* chosen as **v7schema.nickrmt**. **myschema.rmttable** resides in a remote database **rmtdb**, where **rmtdb** has the following characteristics:

- **rmtdb** is a database on a DB2 UDB V7 system (distributed platform.)
- **rmtdb** resides on a remote machine named **rmtsystm** that is accessible through TCP/IP port **34567**.
- **rmtdb** can be accessed using the userid **rmtuser** and password **rmtpwd**.

The following steps should be taken:

1. Verify that the FEDERATED value of the DATABASE MANAGER CONFIGURATION is set to YES. This can be accomplished through the following CLP command (note that the instance must be stopped and restarted in order to have the change take effect):

   ```
   UPDATE DATABASE MANAGER CONFIGURATION USING FEDERATED YES
   ```

2. Verify that the DB2COMM registry variable is set to TCPIP. This can be accomplished through the following system command (note that the instance must be stopped and restarted in order to have the change take effect):

   ```
   db2set DB2COMM=TCPIP
   ```

3. CATALOG a node against the remote system. For this example, the node will be called **distnode**. CLP command:

   ```
   CATALOG TCPIP NODE distnode REMOTE rmtsystm SERVER 34567
   ```

4. If the Federated Database **fedv7** has not been created yet, then create it. CLP command:

   ```
   CREATE DATABASE fedv7
   ```

5. Connect to **fedv7**. CLP command:

   ```
   CONNECT TO fedv7
   ```

6. Create a WRAPPER that corresponds with the remote system. In this case, the correct wrapper to choose is **DRDA**. CLP command:

   ```
   CREATE WRAPPER DRDA
   ```

7. Create a SERVER against the remote system. For this example, the server will be called **distsrvr**. CLP command:

   ```
   CREATE SERVER distsrvr TYPE DB2/UDB VERSION 7.0.0 WRAPPER DRDA AUTHORIZATION
   "rmtuser" PASSWORD "rmtpwd" OPTIONS (NODE 'distnode', DBNAME 'RMTDB')
   ```

   Important: It is recommended that the DBNAME value (in this case, **rmtdb**) be specified in uppercase.

   Important: The meaning of the DBNAME value in the CREATE SERVER command differs between version 7 and Version 8. When creating a SERVER for a version 7 Federated Database, the DBNAME refers to the name of the database on the *remote* system. That database does *not* have to be CATALOGed locally before creating the SERVER.

8. Create a USER MAPPING for **v7user** against **distsrvr**. CLP command:

   ```
   CREATE USER MAPPING FOR v7user SERVER distsrvr OPTIONS (REMOTE_AUTHID 'rmtuser',
   REMOTE_PASSWORD 'rmtpwd')
   ```

9. Create a NICKNAME for **myschema.rmttable**. CLP command:

```
CREATE NICKNAME v7schema.nickrmt FOR distsrvr.myschema.rmttable
```

> **Note:** In this example, **distsrvr.myschema.rmttable** refers to table "**myschema.rmttable**" that resides in the SERVER "**distsrvr**". To create additional NICKNAMEs against the same server (meaning the same remote database), the **distsrvr** SERVER can be used again (additional SERVERs do not need to be created.)

A.1.1.2 Against a mainframe environment

The process of setting up a Federated Database against a mainframe environment is almost identical to that of setting one up against a distributed environment. Please refer to Appendix A.1.1.1, "Against a distributed environment" on page 301. The only changes to the example would be:

- **rmtdb** is a database on a mainframe machine **rmtsystm**, running DB2 for z/OS and OS/390 Version 7.
- The CREATE SERVER command is slightly different (its TYPE is **DB2/MVS** instead of **DB2/UDB**):

```
CREATE SERVER distsrvr TYPE DB2/MVS VERSION 7.0.0 WRAPPER DRDA AUTHORIZATION
"rmtuser" PASSWORD "rmtpwd" OPTIONS (NODE 'distnode', DBNAME 'RMTDB')
```

A.1.2 Federated Database setup with DB2 V8

The process of setting up a Version 8 Federated Database is very similar to the Version 7 process. They differ mainly in the way the CREATE SERVER command is used. While for Version 7, the DBNAME value refers to the name of the database on the remote system, for Version 8, the database on the remote system must first be CATALOGed locally, and then DBNAME value must refer to the locally CATALOGed database. Please refer to the following examples for details:

A.1.2.1 Against a distributed environment

Suppose an instance user **v8user** wishes to set up a Federated Database **fedv8** to be able to access a table **myschema.rmttable** through a *nickname* chosen as **v8schema.nickrmt**. **myschema.rmttable** resides in a remote database **rmtdb**, where **rmtdb** has the following characteristics:

- **rmtdb** is a database on a DB2 UDB V7 system (distributed platform.)
- **rmtdb** resides on a remote machine named **rmtsystm** that is accessible through TCP/IP port **34567**.
- **rmtdb** can be accessed using the userid **rmtuser** and password **rmtpwd**.

The following steps should be taken:

1. Verify that the FEDERATED value of the DATABASE MANAGER CONFIGURATION is set to YES. This can be accomplished through the following CLP command (note that the instance must be stopped and restarted in order to have the change take effect):

```
UPDATE DATABASE MANAGER CONFIGURATION USING FEDERATED YES
```

2. Verify that the DB2COMM registry variable is set to TCPIP. This can be accomplished through the following system command (note that the instance must be stopped and restarted in order to have the change take effect):

```
db2set DB2COMM=TCPIP
```

3. CATALOG a node against the remote system. For this example, the node will be called **distnode**. CLP command:

```
CATALOG TCPIP NODE distnode REMOTE rmtsystm SERVER 34567
```

4. CATALOG the remote database against the node. For this example, the database will be CATALOGed locally as **rmtdblcl**. CLP command:

```
CATALOG DATABASE rmtdb AS rmtdblcl AT NODE distnode
```

> **Note:** This step is only necessary for Version 8 Federated Databases.

5. If the Federated Database **fedv8** has not been created yet, then create it. CLP command:

```
CREATE DATABASE fedv8
```

6. Connect to **fedv8**. CLP command:

```
CONNECT TO fedv8
```

7. Create a WRAPPER that corresponds with the remote system. In this case, the correct wrapper to choose is **DRDA**. CLP command:

```
CREATE WRAPPER DRDA
```

8. Create a SERVER against the remote system. For this example, the server will be called **distsrvr**. CLP command:

```
CREATE SERVER distsrvr TYPE DB2/UDB VERSION 7.0.0 WRAPPER DRDA AUTHORIZATION
"rmtuser" PASSWORD "rmtpwd" OPTIONS (NODE 'distnode', DBNAME 'RMTDBLCL')
```

> **Important:** It is recommended that the DBNAME value (in this case, **rmtdblcl**) be specified in uppercase.

> **Important:** The meaning of the DBNAME value in the CREATE SERVER command differs between version 7 and Version 8. When creating a SERVER for a Version 8 Federated Database, the DBNAME refers to a *locally* CATALOGed database. This means that remote databases *must* first be CATALOGed locally before creating the SERVER.

9. Create a USER MAPPING for **v8user** against **distsrvr**. CLP command:

```
CREATE USER MAPPING FOR v8user SERVER distsrvr OPTIONS (REMOTE_AUTHID 'rmtuser',
REMOTE_PASSWORD 'rmtpwd')
```

10. Create a NICKNAME for **myschema.rmttable**. CLP command:

```
CREATE NICKNAME v8schema.nickrmt FOR distsrvr.myschema.rmttable
```

> **Note:** In this example, **distsrvr.myschema.rmttable** refers to table "**myschema.rmttable**" that resides in the SERVER "**distsrvr**". To create additional NICKNAMEs against the same server (meaning the same remote database), the **distsrvr** SERVER can be used again (additional SERVERs do not need to be created.)

A.1.2.2 Against a mainframe environment

The process of setting up a Federated Database against a mainframe environment is almost identical to that of setting one up against a distributed environment. Please refer to Appendix A.1.2.1, "Against a distributed environment" on page 302. The only changes to the example would be:

- **rmtdb** is a database on a mainframe machine **rmtsystm**, running DB2 for z/OS and OS/390 Version 7.
- CATALOGing the **rmtdb** database from the mainframe system requires that a DCS database be CATALOGed against **rmtdb**, and then that the DCS database be CATALOGed locally against the remote node (in this case, **distnode**). For this example, the DCS database will be called **rmtdbdcs**:

```
CATALOG DCS DATABASE rmtdbdcs AS rmtdb
CATALOG DATABASE rmtdbdcs AS rmtdblcl AT NODE distnode AUTHENTICATION DCS
```

- The CREATE SERVER command is slightly different (its TYPE is **DB2/MVS** instead of **DB2/UDB**):

```
CREATE SERVER distsrvr TYPE DB2/MVS VERSION 7.0.0 WRAPPER DRDA AUTHORIZATION
"rmtuser" PASSWORD "rmtpwd" OPTIONS (NODE 'distnode', DBNAME 'RMTDBLCL')
```

A.2 Server and wrapper type

Server types indicate what kind of data source the server will represent. Server types vary by vendor, purpose, and platform. Supported values depend on the wrapper being used.

DRDA wrapper - DB2 Family

Table A-1 IBM DB2 Universal Database

Server type	Data Source
DB2/UDB	IBM DB2 Universal Database
DataJoiner	IBM DB2 DataJoiner V2.1 and V.2.1.1
DB2/6000	IBM DB2 for UNIX
DB2/HPUX	IBM DB2 for HP-UX V1.2
DB2/NT	IBM DB2 for Windows NT
DB2/EEE	IBM DB2 Enterprise-Extended Edition
DB2/SUN	IBM DB2 for Solaris V1 and V1.2
DB2/2	IBM DB2 for OS/2
DB2/Linux	IBM DB2 for Linux
DB2/PTX	IBM DB2 for NUMA-Q
DB2/SCO	IBM DB2 for SCO Unixware

Table A-2 IBM DB2 Universal Database for AS/400

Server type	Data source
DB2/400	IBM DB2 for AS/400

Table A-3 IBM DB2 Universal Database for OS/390

Server type	Data source
DB2/390	IBM DB2 for OS/390
DB2/MVS	IBM DB2 for MVS

Table A-4 IBM DB2 Server for VM and VSE

Server type	Data source
DB2/VM	IBM DB2 for VM
DB2/VSE	IBM DB2 for VSE
DB2/DS	IBM SQL/DS

SQLNET wrapper

Table A-5 Oracle data sources supported by Oracle SQL*Net V1 or V2 client software

Server type	Data source
ORACLE	Oracle V7.0.13 or later

NET8 wrapper

Table A-6 Oracle data sources supported by Oracle Net8 client software

Server type	Data source
ORACLE	Oracle V7.0.13 or later

Other wrappers

Please consult wrapper documentation.

B

DB2 connectivity

In a distributed data environment, DB2 applications can access data at many different DB2 sites and at remote relational database systems.

A company's distributed environment relies on the distributed data facility (DDF), which is part of DB2 for OS/390 and z/OS. DB2 applications can use *DDF* to access data at other DB2 sites and at remote relational database systems that support Distributed Relational Database Architecture (DRDA). DRDA is a standard for distributed connectivity. All IBM DB2 servers support this DRDA standard.

DDF also enables applications that run in a remote environment that supports DRDA. These applications can use *DB2 Connect* and DDF to access data in DB2 servers on z/OS. DB2 Connect provides connectivity to mainframe or AS/400 from Windows, OS/2, and UNIX-based platforms. You can connect to DB2 databases on AS/400, VSE, VM, MVS, OS/390 and z/OS. You can also connect to non-IBM databases that comply with the Distributed Relational Database Architecture (DRDA). DB2 Connect has several connection solutions:

- Personal Edition, which provides direct connectivity to host or AS/400 databases
- Enterprise Edition, which provides indirect connectivity that allows clients to access host or AS/400 databases through the DB2 Connect server

This appendix describes the minimum customizing the redbook team had to do:

- Communication database on DB2 for z/OS
- Cataloging the databases on DB2 distributed

Figure B-1 shows how the mainframe and the distributed environment is connected.

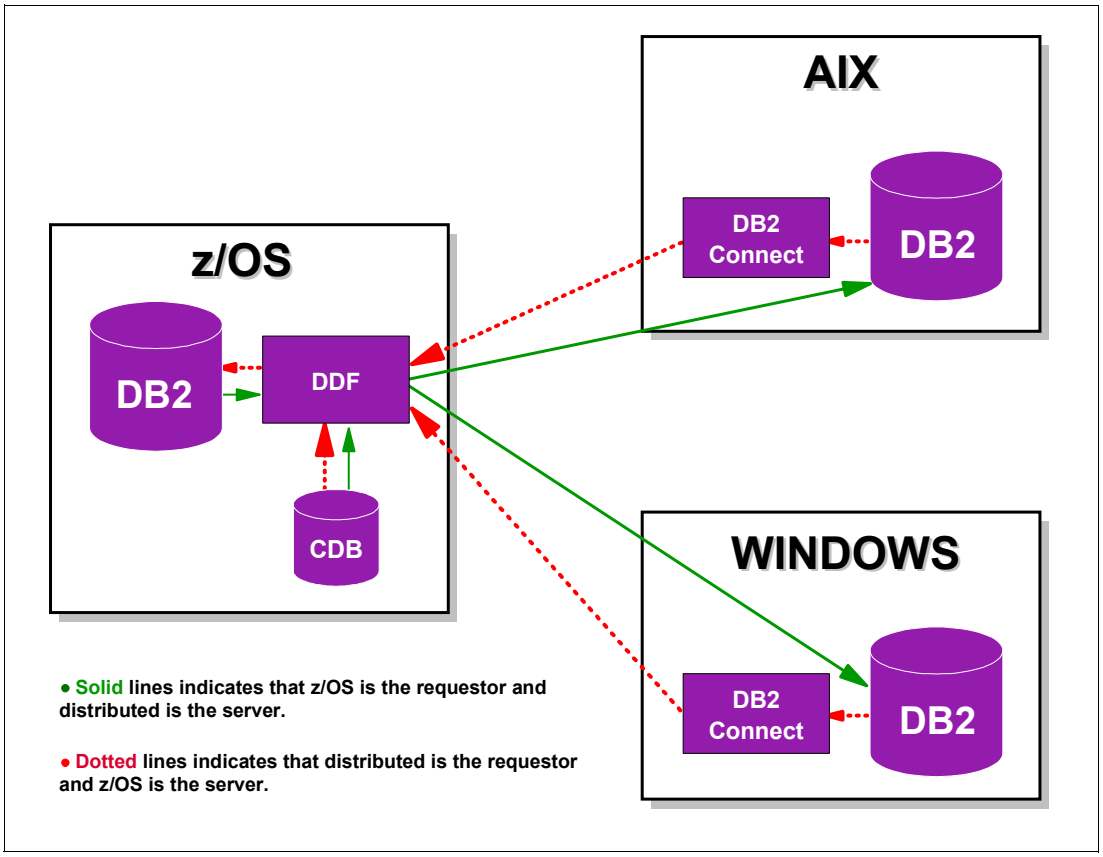

Figure B-1 A distributed data environment

When z/OS is the requestor, DDF will use the location part of the 3.part name to find correct IP-address and portno to the server in its own CDB for the *outgoing* requests. This includes user name and password as well:

```
SELECT * FROM SAMPAIX.DB2INST1.EMP
```

When distributed is the requestor, the DB2 subsystem on z/OS must be cataloged as a database in the distributed system.The DDF verifies the *incoming* requests against the SYSIBM.LUNAMES table in the CDB. If it is not a row with blank in the LUNAME column, DB2 rejects client connections that do not explicitly state a valid LUNAME.

B.1 Communication database on DB2 for z/OS

The DDF uses the CDB to verify in- and outbound requests.

When sending a request from z/OS, DB2 uses the LINKNAME column of the SYSIBM.LOCATIONS table to determine which protocol to use. If the value in the LINKNAME column is found in the:

- SYSIBM.IPNAMES table, TCP/IP is used for DRDA connections
- SYSIBM.LUNAMES table, SNA is used

If the same name is in both SYSIBM.LUNAMES and SYSIBM.IPNAMES, TCP/IP is used to connect to the location.

The table SYSIBM.LUNAMES defines the security and mode requirements for conversations with other systems. Decisions about how to populate this table depend on how you intend to use DB2:

- If you use this system only as a server, DB2 can use a blank in the LUNAME column as a default.
- If this DB2 requests data from other systems, you need to provide LU names or IP names for those systems.

If you do not have a row with a blank in the LUNAME column, DB2 rejects client connections that do not explicitly state a valid LUNAME.

For details, see *DB2 Universal Database for OS/390 and z/OS Installation Guide*, GC26-9936-02, and *DB2 Universal Database for OS/390 and z/OS Administration Guide*, SC26-9931-02.

B.1.1 Populate the communication database

If you plan to use DB2 on the z/OS only as a server, you do not need to populate the CDB. However, if you intend to request data, you need to enter rows in the tables:

- SYSIBM.LOCATIONS
- SYSIBM.IPNAMES
- SYSIBM.USERNAMES

Note: SYSIBM.LUNAMES must at least contain the default row inserted in the DSNTIJSG installation job.

B.1.1.1 SYSIBM.LOCATIONS table

The table LOCATIONS is used to determine the port number or service name used to connect to the remote location. The column LINKNAME maps to the corresponding row in table IPNAMES. Insert statement, see Example B-1.

LOACTIONS has the following columns:

LOCATION The unique network location name, or DRDA RDBNAM, assigned to a system, remote or local. You must provide location names for any systems that you request data from. This column is the primary key for this table.

LINKNAME Identifies the TCP/IP attributes associated with this location. For each LINKNAME specified, you must have a row in SYSIBM.IPNAMES whose LINKNAME matches the value specified in this column.

Because this table is used for outbound requests, you must provide a LINKNAME or your requests fail. Do not enter blanks in this column.

PORT The port number of the remote database server. The number must be 1 to 5 characters and left justified.

Example: B-1 Insert into LOCATIONS table

```
INSERT INTO SYSIBM.LOCATIONS (LOCATION, LINKNAME, PORT)
       VALUES ('SAMPAIX','DMAIX', '50000');
INSERT INTO SYSIBM.LOCATIONS (LOCATION, LINKNAME, PORT)
       VALUES ('SAMPWIN','DMWIN', '50000');
```

Restriction: Column LOCATION is the primary key in the LOCATIONS table, which means you have to create alias on databases with the same name.

B.1.1.2 SYSIBM.IPNAMES table

IPNAMES defines the outbound security and host names used to connect to other systems using TCP/IP. Insert statement; see Example B-2.

IPNAMES has the following columns:

LINKNAME This value matches that specified in the LINKNAME column of the associated row in SYSIBM.LOCATIONS.

SECURITY_OUT Defines the security option that is used when local DB2 SQL applications connect to any remote server associated with this TCP/IP host. The default, A, means that outgoing connection requests contain an authorization ID without a password.

USERNAMES This column is used for outbound requests to control translations of authorization IDs. The values 'O' or 'B' are valid for TCP/IP connections.

IPADDR This column contains the IP address or domain name of a remote TCP/IP host. An IP address must be 1 to 15 characters and left justified.

Example: B-2 Insert into IPNAMES table

```
INSERT INTO SYSIBM.IPNAMES (LINKNAME, SECURITY_OUT, USERNAMES, IPADDR)
       VALUES ('DMAIX', 'P', 'O', '9.1.38.177');
INSERT INTO SYSIBM.IPNAMES (LINKNAME, SECURITY_OUT, USERNAMES, IPADDR)
       VALUES ('DMWIN', 'P', 'O', '9.1.39.42');
```

B.1.1.3 SYSIBM.USERNAMES table

USERNAMES contains information needed for outbound translation only. Reminder: Inbound ID translation and come from checking are not done for TCP/IP requesters. Insert statement, see Example B-3.

USERNAMES has the following columns:

TYPE Whether the row is for outbound translation. The value 'O' is valid for TCP/IP connections.

AUTHID Authorization ID to translate. If blank, it applies to all authorization IDs.

LINKNAME Identifies the TCP/IP network location associated with the row. A blank indicates it applies to all TCP/IP partners. For non blank values, this value must match the LINKNAME value in SYSIBM.IPNAMES.

NEWAUTHID The translated value of AUTHID.

PASSWORD The password to accompany an outbound request. This column is ignored if RACF PassTickets, or already verified USERIDs are used.

Example: B-3 Insert into USERNAMES table

```
INSERT INTO SYSIBM.USERNAMES (TYPE, LINKNAME, NEWAUTHID, PASSWORD)
       VALUES ('O', 'DMAIX', 'db2inst1', 'xxxxxxxx');
INSERT INTO SYSIBM.USERNAMES (TYPE, LINKNAME, NEWAUTHID, PASSWORD)
       VALUES ('O', 'DMWIN', 'db2admin', 'xxxxxxxx');
```

B.1.2 CDB tables with contents

After inserting the rows described in Chapter B.1.1, “Populate the communication database” on page 309 the content should look like Example B-4:

Example: B-4 Rows in CDB tables

SYSIBM.LOCATIONS

```
LOCATION          LINKNAME  IBMREQD  PORT                              TPN
---------+---------+---------+---------+---------+---------+---------+---------+
SAMPAIX           DMAIX     N        50000
SAMPWIN           DMWIN     N        50000
```

SYSIBM.IPNAMES

```
LINKNAME  SECURITY_OUT  USERNAMES  IBMREQD  IPADDR
---------+---------+---------+---------+---------+---------+---------+---------+
DMAIX     P             O          N        9.1.38.177
DMWIN     P             O          N        9.1.39.42
```

SYSIBM.USERNAMES

```
TYPE  AUTHID    LINKNAME  NEWAUTHID  PASSWORD  IBMREQD
---------+---------+---------+---------+---------+---------+---------+---------+
O               DMAIX     db2inst1   xxxxxxxx N
O               DMWIN     db2admin   xxxxxxxx N
```

You should ensure that the LUNAMES table contains at least the row shown in Example B-5 and that the LUNAME is blank:

Example: B-5 Rows in LUNAMES table

```
SYSIBM.LUNAMES

LUNAME  SYSMODENAME SECURITY_IN SECURITY_OUT ENCRYPTPSWDS MODESELECT USERNAMES GENERIC IBMR
-------+--------+--------+--------+---------+--------+--------+---------+----------+-------
                   A          A            N           N                   N       N
```

B.1.3 Test the connectivity

Important: You have to stop and start the DB2 DDF address space to make the changes in the CDB tables effective. If not, this error occurs when you try a select:

```
SQLCODE = -30061, ERROR:  RDB NOT FOUND
```

From SPUFI you can try the following select:

```
SELECT * FROM SAMPAIX.DB2INST1.DEPARTMENT ORDER BY 1;
```

If you get a SQL -805 on package DSNESM68 shown in Example B-6, you have to bind it shown in Example B-7.

Example: B-6 SQL -805 in SPUFI

```
DSNT408I SQLCODE =   -805, SQLSTATE = 51002, INVALID APPLICATION STATE FROM
         OS/2     TOKENS DSNESPCS.DSNESM68
```

Example: B-7 Bind SPUFI cursor stability plan

```
BIND PACKAGE (SAMPAIX.DSNESPCS) MEMBER(DSNESM68) -
                    LIB('DB2G7.SDSNDBRM')          -
                    ISOLATION(CS)                  -
                    ACTION(REPLACE)
```

Be aware of the warnings you get in the bind result, see Example B-8:

Example: B-8 Result from the bind

```
DSN
  BIND PACKAGE (SAMPAIX.DSNESPCS) MEMBER(DSNESM68)
               LIB('DB2G7.SDSNDBRM')                    ISOLATION(CS)
WARNING, ONLY IBM-SUPPLIED COLLECTION-IDS SHOULD BEGIN WITH  "DSN"
WARNING, ONLY IBM-SUPPLIED PACKAGE-IDS SHOULD BEGIN WITH  "DSN"
DSNT232I  -DB2G SUCCESSFUL BIND FOR
           PACKAGE = SAMPAIX.DSNESPCS.DSNESM68.()
DSN
```

Note: If you are using repeatable read, you also have to bind the DSNESPRR plan qualified with the same location name.

Example B-9 shows the result from a SQL statement where the location name is not defined in the CDB:

```
SELECT * FROM AIXSAMP.DB2INST1.STAFF
```

Example: B-9 The location name of the remote site is not defined in the CDB

```
DSNT408I SQLCODE = -950, ERROR:  THE LOCATION NAME SPECIFIED IN THE CONNECT
         STATEMENT IS INVALID OR NOT LISTED IN THE COMMUNICATIONS DATABASE
```

Example B-10 shows the result set from a successful execution in SPUFI:

Example: B-10 Result set from the DEPARTMENT table on AIX in SPUFI

```
******************************** Top of Data ********************************
---------+---------+---------+---------+---------+---------+---------+---------+
SELECT * FROM SAMPAIX.DB2INST1.DEPARTMENT ORDER BY 1;                   00020009
---------+---------+---------+---------+---------+---------+---------+---------+
DEPTNO  DEPTNAME                       MGRNO   ADMRDEPT  LOCATION
---------+---------+---------+---------+---------+---------+---------+---------+
A00     SPIFFY COMPUTER SERVICE DIV.   000010  A00       ----------------
B01     PLANNING                       000020  A00       ----------------
C01     INFORMATION CENTER             000030  A00       ----------------
D01     DEVELOPMENT CENTER             ------  A00       ----------------
D11     MANUFACTURING SYSTEMS          000060  D01       ----------------
D21     ADMINISTRATION SYSTEMS         000070  D01       ----------------
E01     SUPPORT SERVICES               000050  A00       ----------------
E11     OPERATIONS                     000090  E01       ----------------
E21     SOFTWARE SUPPORT               000100  E01       ----------------
DSNE610I NUMBER OF ROWS DISPLAYED IS 9
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
---------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
```

Note: To activate the connection to the database on windows, you have to bind the DSNESPCS and DSNESPRR plans qualified with the actual location name. In this scenario, it means SAMPWIN.

B.2 Cataloging the databases on DB2 distributed

In the following, there are screen shots to show how do define the connection to the mainframe to be used by DB2 Connect.

The easiest way to set up the mainframe connection is to use the Client Configure Assistant. DB2 have to be installed and activate the wizard like this:

```
start menu ==> Program ==> IBM DB2 ==> Client Configuration Assistant
```

During this project we used DB2 subsystem DB2G on the mainframe in Poughkeepsie. Figure B-2 shows the first window and the databases that are configured. Push the **ADD** button.

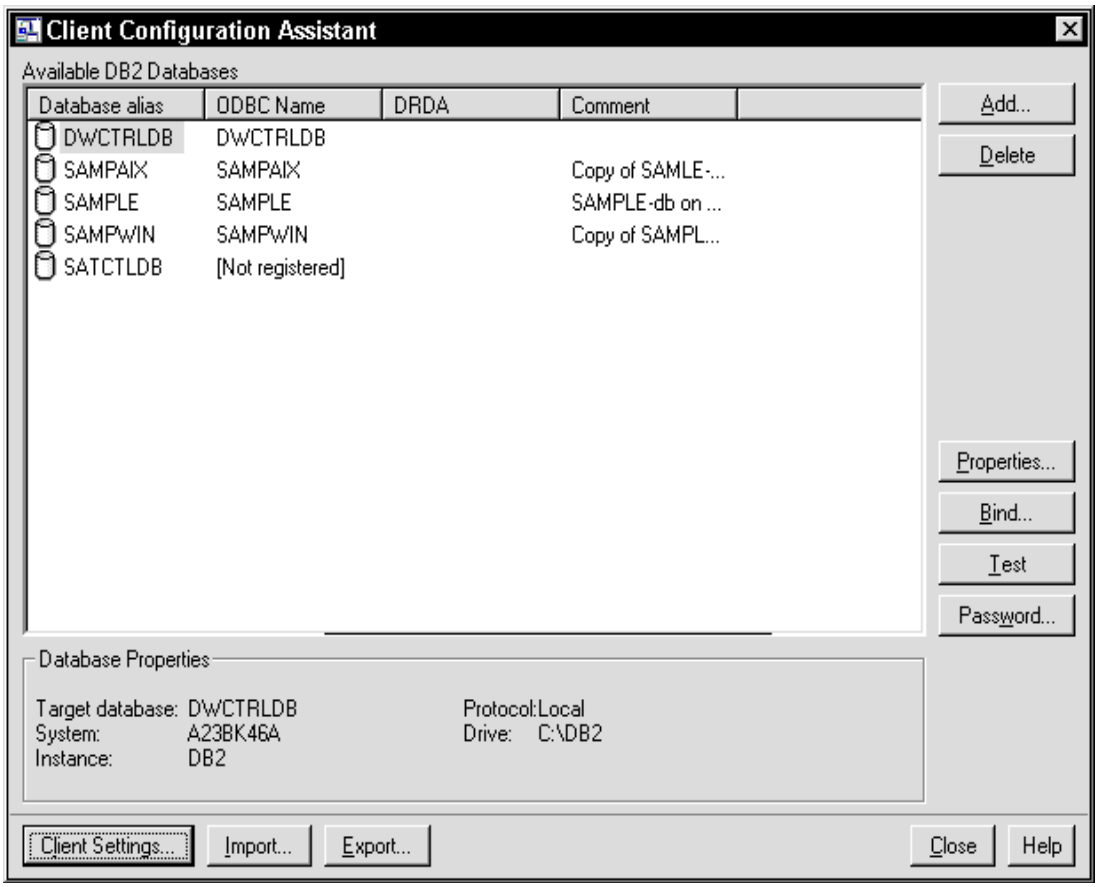

Figure B-2 Client Configuration Assistant — Available DB2 databases

If the IP-address is known, choose **Manually** In Figure B-3 and press the **Next** button.

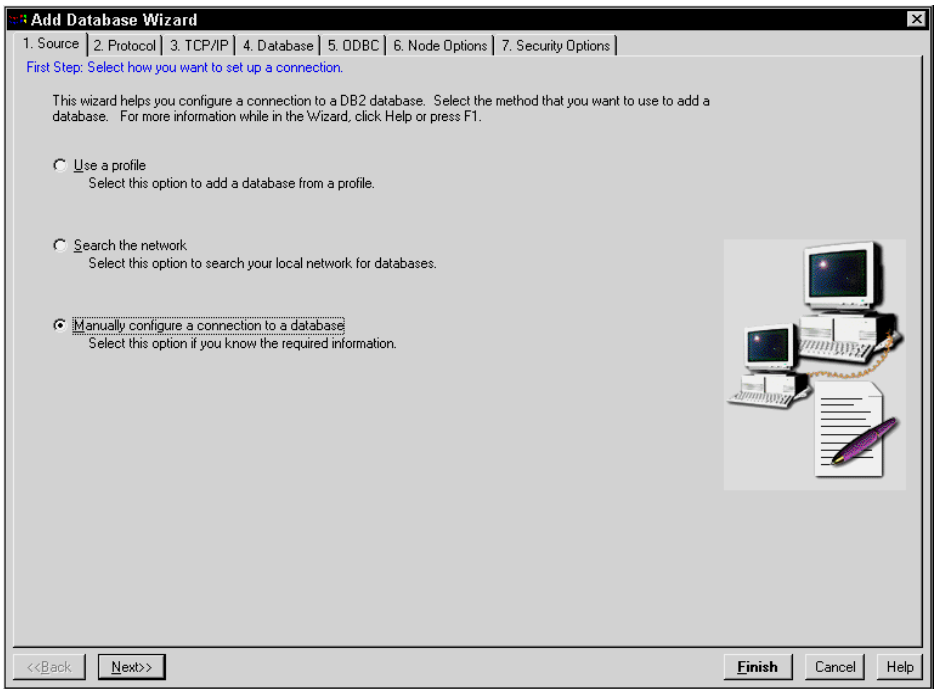

Figure B-3 Client Configuration Assistant 1 — Source

Choose options as shown In Figure B-4 and press the **Next** button.

.

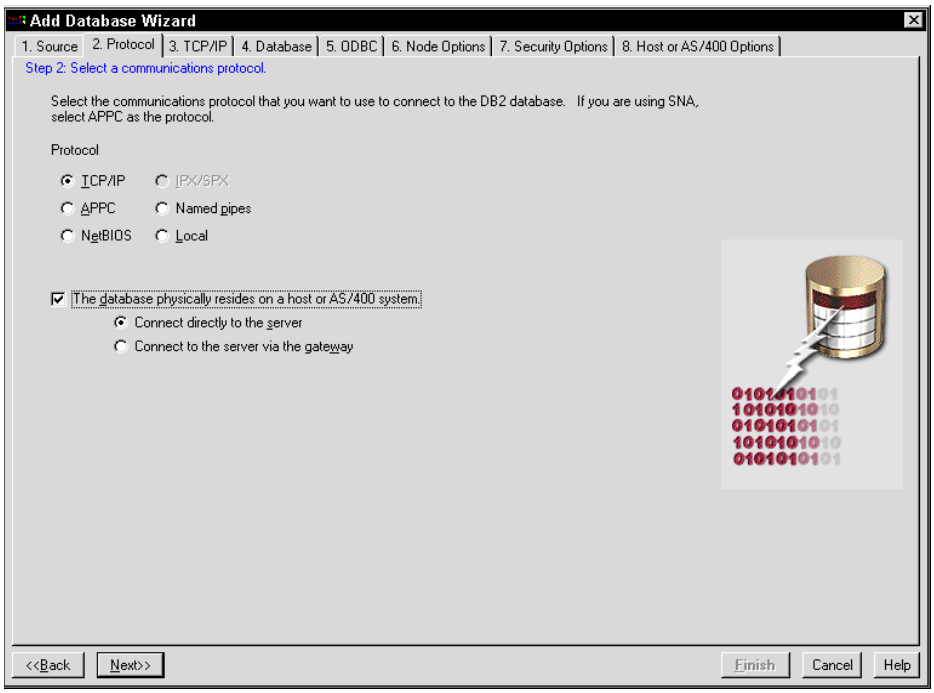

Figure B-4 Client Configuration Assistant 2 — Source

Type the IP-address and the Port number to the mainframe as shown In Figure B-5 and press the **Next** button.

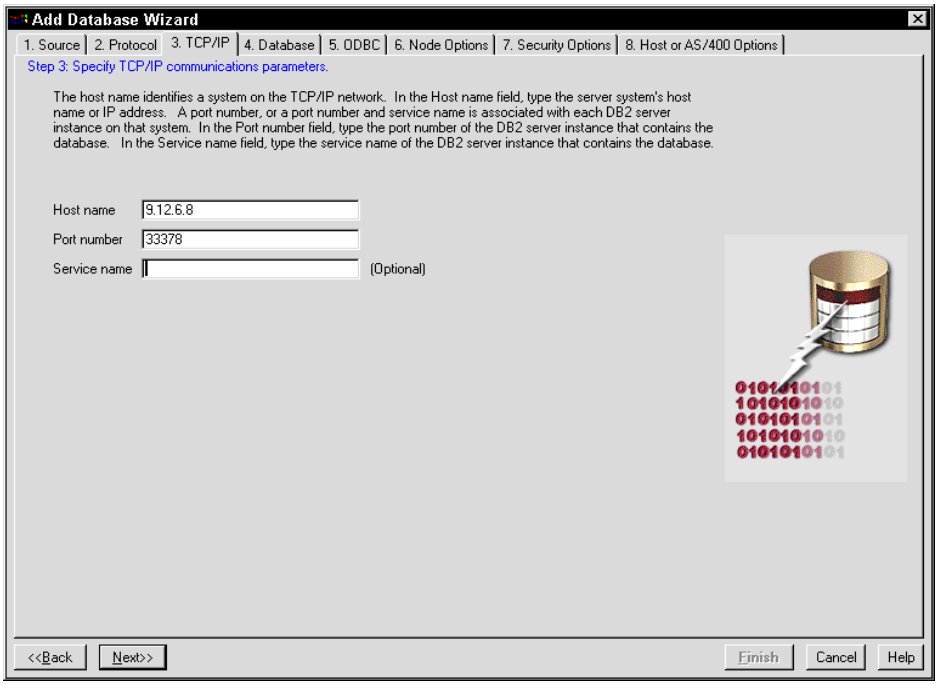

Figure B-5 Client Configuration Assistant 3 — TCP/IP

Type the subsystem name on the mainframe as shown In Figure B-6 and press the **Next** button. Use the comment field for more details.

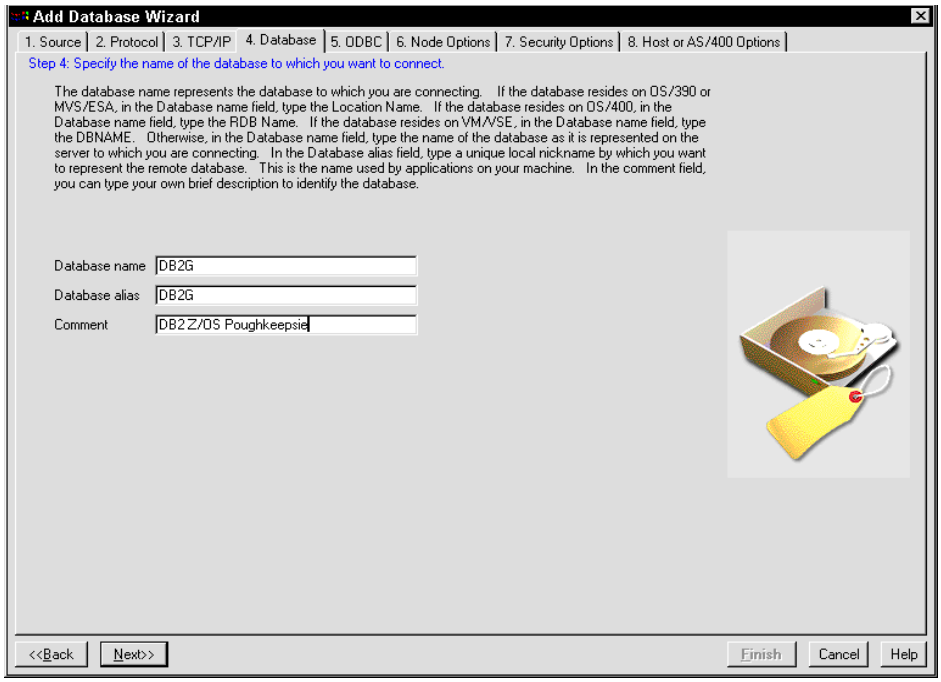

Figure B-6 Client Configuration Assistant 4 — Database

The following 4 windows are optional and we did not fill them out:

- 5 — ODBC
- 6 — Node options
- 7 — Security options
- 8 — Host options

After the eight windows, test the connection as shown In Figure B-7. When you get `The connection test was successful` click the **OK** button. Hopefully, the mainframe connection is added to list of available databases in Figure B-2.

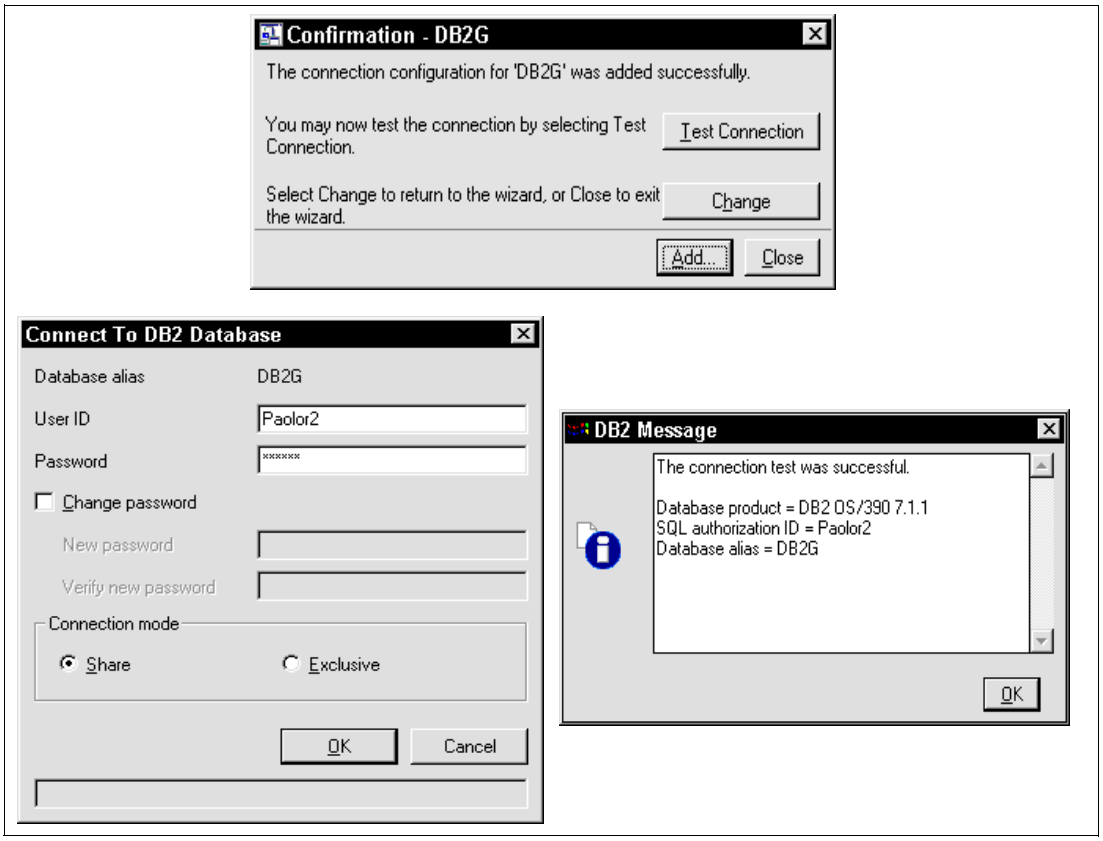

Figure B-7 Client Configuration Assistant — Test connection

The equivalent test connection CLP script to the sequence of panels in Figure B-7 is listed in Figure B-11.

Example: B-11 Test connection CLP script

```
catalog tcpip node hostnode remote 9.12.6.8 server 33378;
catalog DB2G as DB2G at node hostnode authentication dcs:
connect to DB2G user paolor2 using xxxxxx;
```

C

Migrating to DB2 distributed V8

Moving data across the DB2 Family sometimes also includes migrating to a new DB2 version. Migrating your database is data movement that involves moving not just individual tables, but also the DDL used to create the tables, and other table definitions like referential integrity. Planning is essential *before* you do your database migration. It is prudent to have an overview of all the steps that you are going to make before you actually start. Failure to do so may require the whole operation to be restarted due to error.

Although there are also some post-operation tasks that should be done in single table data movements like Unload and Load operation. We will discuss these tasks on the chapter that refer specifically to that tool or utility.

C.1 Migration restrictions

There are certain pre-conditions or restrictions that you should be aware of before attempting to migrate your database.

Migration is only supported from DB2 UDB V6 and later releases, and also Data Joiner V2.1.1. Migration from DB2 V1.2. Parallel Edition is not supported. Earlier versions of DB2 (Database Manager) must be migrated to V5.x or V6 before being migrated to a later release.

1. The migration command has to be issued from a client level corresponding to the release level of the new database server. Migration from a down-level client is not supported.
2. Migration between platforms is not supported.
3. User objects within your database cannot have object names identical to reserved schema names existing in the new release. These reserved schema names include: SYSCAT, SYSSTAT and SYSFUN.
4. User-defined distinct types using the names BIGINT, REAL, DATALINK, or REFERENCE must be renamed before migrating the database.
5. You cannot migrate a database that is in one of the following states:
   - Backup pending
   - Roll-forward pending
   - One or more table spaces not in a normal state
   - Transaction inconsistent

Restoration of down-level database backups is supported, but the rolling forward of down-level logs is not supported.

C.2 Pre- and post-migration tasks

Following are the steps you must take to migrate your database to DB2 version 8. The database manager must be started before migration can begin.

Pre-migration tasks

The pre-migration steps must be done on a previous release (that is, on your current release before migrating to, or installing, the new release.)

1. Verify that there are no unresolved issues that pertain to “Migration restrictions” section.
2. Disconnect all applications and end users from each database being migrated (use the LIST APPLICATIONS command, or the FORCE APPLICATIONS command, as necessary.)
3. Use the DB2CKMIG pre-migration utility to determine if the database can be migrated (for detailed information about using this utility, see the *Quick Beginnings* book for your platform). Note that on Windows NT or OS/2, you are prompted to run this tool during installation, but on UNIX based systems, this tool is invoked automatically during instance migration.
4. Back up your database.

Migration is not a recoverable process. If you back up your database before the Version 6 reserved schema names are changed, you will not be able to restore the database using DB2 UDB Version 8. To restore the database, you will have to use your previous version of the database manager.

Attention: If you do not have a backup of your database, and the migration fails, you will have no way of restoring your database using DB2 UDB Version 8, or your previous version of the database manager.

You should also be aware that any database transactions done between the time the backup was taken and the time that the upgrade to Version 8 is completed are not recoverable. That is, if at some time following the completion of the installation and migration to Version 8, the database needs to be restored (to a Version 8 level), the logs written before Version 8installation cannot be used in roll-forward recovery.

Post-migration tasks on DB2 Version 8

1. Optionally, use the DB2UIDDL utility to facilitate the management of a staged migration of unique indexes on your own schedule. (DB2 Version 5 databases that were created in Version 5 do not require this tool to take advantage of deferred uniqueness checking, because all unique indexes created in Version 5 have these semantics already. However, for databases that were previously migrated to Version 5, these semantics are not automatic, unless you use the DB2UIDDL utility to change the unique indexes.) This utility generates CREATE UNIQUE INDEX statements for unique indexes on user tables, and writes them to a file. Running this file as a DB2 CLP command file results in the unique index being converted to Version 8 semantics. For detailed information about using this utility, refer to one of the *Quick Beginnings* books.
2. Optionally, issue the RUNSTATS command against tables that are particularly critical to the performance of SQL queries. Old statistics are retained in the migrated database, and are not updated unless you invoke the RUNSTATS command.
3. Optionally, use the DB2RBIND utility to revalidate all packages, or allow package revalidation to occur implicitly when a package is first used.
4. Optionally, migrate Explain tables if you are planning to use them in Version 8. For more information, see the SQL Explain Facility in the *Administration Guide: Performanc*e.
5. Tune your database and database manager configuration parameters to take advantage of Version 8 enhancements.

D

DB2 UDB for z/OS Unload options

This appendix can be used as references while choosing the Unload options. More information in *DB2 UDB for OS/390 and z/OS Version 7 Utility Guide and Reference,* SC26-9945-03.

DATA

Identifies the data selected for unloading with table-name in the from-table-spec. The DATA keyword is mutually exclusive with TABLESPACE, PART and LIST keywords and from-copy-spec.

When you specify the DATA keyword, or you omit either the TABLESPACE or the LIST keyword, you must also specify at least one FROM TABLE clause.

TABLESPACE

Specifies the table space (and, optionally, the database to | which it belongs) from which the data is unloaded.

database-name

The name of the database to which the table space belongs. The name cannot be DSNDB01 or DSNDB07.

The default is DSNDB04.

tablespace-name

The name of the table space from which the data is unloaded. The specified table space must not be a LOB table space.

PART

Identifies a partition or a range of partitions from which the data is unloaded. This keyword applies when the specified table space is partitioned; it cannot be specified with LIST. The maximum is 254.

integer
Designates a single partition. integer must be an existing partition number within the table space.

int1:int2
Designates a range of partitions from int1 to int2. int1 must be a positive integer that is less than the highest partition number within the table space. int2 must be an integer greater than int1 and less than or equal to the highest partition number.

If no PART keyword is specified in an UNLOAD control statement, the data from the entire table space will be unloaded into a single unload data set.

FROMCOPY data-set-name
Indicates that data is unloaded from an image copy data set. When you specify FROMCOPY, the Unload utility processes only the specified image copy data set. Alternatively, the FROMCOPYDDN keyword can be used where multiple image copy data sets can be concatenated under a single DD name.

data-set-name
Specifies the name of a single image copy data set.

The FROMCOPY image copy data set must have been created by one of the following utilities:

- COPY
- # COPYTOCOPY
- LOAD inline image copy
- MERGECOPY
- REORG TABLESPACE inline image copy
- DSN1COPY

If the specified image copy data set is a full image copy, either compressed or uncompressed records can be unloaded.

If the specified image copy data set is an incremental image copy or a copy of a partition or partitions, you can unload compressed records only when the same data set contains the dictionary pages for decompression. If an image copy data set contains a compressed row and a dictionary is not available, then DB2 issues an error message.

When you specify FROMCOPY or FROMCOPYDDN, you can also specify selection criteria with either PART, or FROM TABLE, or both, to qualify tables and rows to be unloaded.

FROMVOLUME
Identifies the image copy data set.

CATALOG
Identifies that the data set is cataloged. Use this option only for an image copy that was created as a cataloged data set (its volume serial is not recorded in SYSIBM.SYSCOPY.)

vol-ser
Identifies the data set by an alphanumeric volume serial identifier of its first volume. Use this option only for an image copy that was created as a non-cataloged data set. To locate a data set stored on multiple tape volumes, specify the first vol-ser in the SYSCOPY record.

FROMCOPYDDN ddname
Indicates that data is unloaded from one or more image copy data sets associated with the specified DDNAME. Multiple image copy data sets (primarily for copy of pieces) can be concatenated under a single DD name.

ddname
Identifies a DD name with which one or more image copy data sets are associated.

LIST listdef-name
Identifies the name of a list of objects that are defined by a LISTDEF utility control statement. Index spaces, LOB table spaces, and directory objects must not be included in the list. You cannot use the LIST option to specify image copy data sets.

When you specify the LIST option, the referenced LISTDEF identifies:

- The table spaces from which the data is unloaded (you can use the pattern-matching feature of LISTDEF.)
- The partitions (if a table space is partitioned) from which the data is unloaded (defined by the INCLUDE, EXCLUDE and PARTLEVEL keywords in the LISTDEF statement.)

The Unload utility associates a single table space with one output data set, except when partition-parallelism is activated. When you use the LIST option with a LISTDEF that represents multiple table spaces, you must also define a data set TEMPLATE that corresponds to all the table spaces and specify the template-name in the UNLDDN option.

If you want to generate the LOAD statements, you must define another TEMPLATE for the PUNCHDDN data set that is similar to UNLDDN. DB2 then generate a LOAD statement for each table space.

PUNCHDDN
Specifies the DD name for a data set, or a template name, that defines one or more data set names to receive the LOAD utility control statements that the Unload utility generates.

ddname
Specifies the DD name. The default is SYSPUNCH.

template-name
Identifies the name of a data set template that is defined by a TEMPLATE utility control statement.

If the name is defined as a DD name by the JCL, it is treated as the DD name.

When you run the Unload utility for multiple table spaces and you want to generate corresponding LOAD statements, you must have multiple output data sets that correspond to the table spaces so that DB2 retains all of the generated LOAD statements. In this case, you must specify an appropriate template name to PUNCHDDN. If you omit the PUNCHDDN specification, the LOAD statements are not generated.

If the partition variable (&PART. or &PA). is included in a TEMPLATE for PUNCHDDN, DB2 replaces the &PART. or &PA. variable with the lowest partition number in the list of partitions to be unloaded. The partition number is in the form nnnnn.

UNLDDN

Specifies the DD name for a data set or a template name that defines one or more data set names into which the data is unloaded.

ddname

Specifies the DD name. The default is SYSREC.

template-name

Identifies the name of a data set template that is defined by a TEMPLATE utility control statement.

If the specified name is defined both as a DDNAME (in the JCL) and as a template name (a TEMPLATE statement), it is treated as the DDNAME.

When you run the Unload utility for a partitioned table space, the selected partitions are unloaded in parallel if a template name is specified in UNLDDN and the template data set name contains the partition as a variable (&PART. or &PA.). In this case, the template is expanded into multiple data sets that correspond to the selected partitions. For example, the data from partitions 2 and 5 of the table space testdb.testtsp1 are unloaded in parallel into two data sets by using the following utility statement:

```
TEMPLATE unldtemp
   DSNAME(&DBNAME..&TSNAME..P&PART.)
LISTDEF unldlist
   INCLUDE TABLESPACE testdb.testtsp1 PARTLEVEL (2)
   INCLUDE TABLESPACE testdb.testtsp1 PARTLEVEL (5)
UNLOAD LIST unldlist UNLDDN unldtempl ...
```

Similarly, when you run the Unload utility for multiple table spaces, the output records are placed in data sets that correspond to the respective table spaces; therefore, the output data sets must be physically distinctive, and you must specify an appropriate template name to UNLDDN. If you omit the UNLDDN specification, the SYSREC DDNAME is not used and an error occurs.

If the partition variable (&PART. or &PA). is included in TEMPLATE DSNAME when the partition parallelism is not applicable (when the source is a non-partitioned table space or an image copy), the variable is replaced by '00000' in the actual data set name. In this case, warning message DSNU 1252 is issued, and the Unload utility issues return code 4.

EBCDIC

Specifies that all output data of the character type will be in EBCDIC. If a different encoding scheme is used for the source data, the data is converted into EBCDIC (except for bit strings.)

If you do not specify either EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved.

See the description of the CCSID option for this utility.

ASCII

Specifies that all output data of the character type will be in ASCII. If a different encoding scheme is used for the source data, the data is converted into ASCII (except for bit strings.)

If you do not specify either EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved.

See the description of the CCSID option for this utility.

UNICODE

Specifies that all output data of the character type will be in UNICODE (except for bit strings.) If a different encoding scheme is used for the source data, the data is converted into UNICODE.

If you do not specify either EBCDIC, ASCII, UNICODE, or CCSID, the encoding scheme of the source data is preserved. See the description of the CCSID option of this utility.

CCSID

Specifies up to three coded character set identifiers (CCSIDs) to be used for the data of character type in the output records, including data unloaded in the external character formats.It is coded as:

```
CCSID(integer1,integer2,integer3)
```

Iinteger1 specifies the CCSID for SBCS data.

- integer2 specifies the CCSID for mixed data.
- integer3 specifies the CCSID for DBCS data. This option is not applied to data with a subtype of BIT.

The following specifications are also valid:

- CCSID(integer1)

  Only an SBCS CCSID is specified.

- CCSID(integer1,integer2)

  An SBCS CCSID and a mixed CCSID are specified.

  – integer

    Valid CCSID specification or 0.

If you specify an argument as 0 or omit an argument, the encoding scheme specified by EBCDIC, ASCII, or UNICODE is assumed for the corresponding data type (SBCS, MIXED, or DBCS). If you do not specify either EBCDIC, ASCII, or UNICODE:

If the source data is of character type, the original encoding scheme is preserved. For character strings that are converted from numeric data, date, time or timestamp, the default encoding scheme of the table is used. For more information, see the CCSID option of the CREATE TABLE statement in Chapter 5 of DB2 SQL Reference.

When a CCSID conversion is requested, CCSID character substitutions can occur in the output string. Use the NOSUBS option to prevent possible character substitutions during CCSID conversion.

NOSUBS

Specifies that CCSID code substitution is not to be performed during unload processing.

When a string is converted from one CCSID to another (including EBCDIC, ASCII, and UNICODE), a substitution character is sometimes placed in the output string. For example, this substitution occurs when a character (referred to as a codepoint) that exists in the source

CCSID does not | exist in the target CCSID. You can use the NOSUBS keyword to | prevent the Unload utility from allowing this substitution.

If you specify the NOSUBS keyword and character substitution is attempted while unloading data, it is treated as a conversion error. The record with the error is not unloaded, and the process continues until the total error count reaches the number specified by MAXERR.

NOPAD

Specifies that the variable length columns in the unloaded records occupy the actual data length without additional padding. As a result, the unloaded or discarded records might have varying lengths.

When you do not specify NOPAD, default UNLOAD processing pads variable length columns in the unloaded records to their maximum length, and the unloaded records have the same length for each table.

The padded data fields are preceded by the length fields that indicate the size of the actual data without the padding.

When the output records are reloaded using the LOAD utility, padded data fields are treated as varying length data.

While LOAD processes records with variable length columns that are unloaded or discarded by using the NOPAD option, these records cannot be processed by applications that only process fields in fixed positions. For example, the LOAD statement generated for the EMP sample table would look similar to the LOAD statement shown.

FLOAT

Specifies the output format of the numeric floating point data. This option applies to the binary output format only.

S390

Indicates that the binary floating point data is written to the output records in the S/390® internal format (also known as the *hexadecimal floating point* or HFP.)

The default is FLOAT S390.

IEEE

Indicates that the binary floating point data is written to the output records in the IEEE format (also known as the *binary floating point* or BFP). The IEEE option is applicable only when OS/390 Version 2 Release 6 or a subsequent version is installed, and a G5 or above processor is present.

MAXERR

Specifies the maximum number of records in error that are allowed; the unloading process terminates when this value is reached. It is coded as:

```
MAXERR integer
```

This value specifies the number of records in error that are allowed. When the error count reaches this number, the Unload utility issues message DSNU1219 and terminates with return code 8.

The default is 1, which indicates that Unload stops when the first error is encountered. If you specify 0 or any negative number, execution continues regardless of the number of records in error.

If multiple table spaces are being processed, the number of records in error is counted for each table space. If the LIST option is used, you can add OPTION utility control statement (EVENT option with ITEMERROR) before the UNLOAD statement to specify that the table space in error is skipped and the subsequent table spaces are processed.

SHRLEVEL

Specifies whether other processes can access or update the table space or partitions while the data is being unloaded.

Unload ignores the SHRLEVEL specification when the source object is an image copy data set. The default is SHRLEVEL CHANGE ISOLATION CS.

CHANGE

Specifies that rows can be read, inserted, updated, and deleted from the table space or partition while the data is being unloaded.

ISOLATION

Specifies the isolation level with SHRLEVEL CHANGE.

CS

Indicates that the Unload utility reads rows in cursor stability mode. With CS, the Unload utility assumes CURRENTDATA(NO.)

UR

Indicates that uncommitted rows, if they exist, are unloaded. The unload operation is performed with minimal interference from the other DB2 operations that are applied to the objects from which the data is being unloaded.

REFERENCE

Specifies that during the unload operation, rows of the tables can be read, but cannot be inserted, updated, nor deleted by other DB2 threads.

When you specify SHRLEVEL REFERENCE, the Unload utility drains writers on the table space from which the data is to be unloaded. When data is unloaded from multiple partitions, the drain lock will be obtained for all of the selected partitions in the UTILINIT phase.

Abbreviations and acronyms

AIX	Advanced Interactive eXecutive from IBM
APAR	authorized program analysis report
AR	access register
ARM	automatic restart manager
ART	access register translation
ASCII	American National Standard Code for Information Interchange
BLOB	binary large object
CCA	client configuration assistant
CCSID	coded character set identifier
CD	compact disk
CEC	central electronics complex
CF	coupling facility
CFCC	coupling facility control code
CFRM	coupling facility resource management
CLI	call level interface
CLOB	character large object
CLP	command line processor
CPU	central processing unit
CRLF	carriage return and line feed
CSA	common storage area
CTT	created temporary table
DAD	document access definition
DASD	direct access storage device
DAT	dynamic address translation
DB2 PM	DB2 performance monitor
DBAT	database access thread
DBCLOB	double byte character large object
DBET	database exception tables states
DBD	database descriptor
DBID	database identifier
DBMS	database management system
DBRM	database request module
DCL	data control language
DDCS	distributed database connection services
DDF	distributed data facility
DDL	data definition language
DEL	delimited ASCII
DLL	dynamic load library manipulation language
DML	data manipulation language
DNS	domain name server
DPSI	data partitioned secondary index
DRDA	distributed relational database architecture
DSC	dynamic statement cache, local or global
DTT	declared temporary tables
EA	extended addressability
EBCDIC	extended binary coded decimal interchange code
ECS	enhanced catalog sharing
ECSA	extended common storage area
EDM	environment descriptor management
ENFM	enabling new function mode
ERP	enterprise resource planning
ESA	Enterprise Systems Architecture
ESS	Enterprise Storage Server
ETR	*e*xternal throughput rate, an elapsed time measure, focuses on system capacity
FDT	functional track directory
FTP	File Transfer Program
GB	gigabyte (1,073,741,824 bytes)
GBP	group buffer pool
GRS	global resource serialization
GUI	graphical user interface
HA	Host adapter
HFS	Hierarchical File System
HPJ	high performance Java
I/O	input/output
IBM	International Business Machines Corporation
ICF	integrated catalog facility
ICF	integrated coupling facility
ICMF	internal coupling migration facility
IFCID	instrumentation facility component identifier
IFI	instrumentation facility interface

IPLA IBM Program Licence Agreement

IRLM internal resource lock manager

IRWW IBM Relational Warehouse Workload

ISPF interactive system productivity facility

ISV independent software vendor

ITR internal throughput rate, a processor time measure, focuses on processor capacity

ITSO International Technical Support Organization

IVP installation verification process

IXF integration exchange format

JDBC Java Database Connectivity

JFS journaled file systems

JIT Just in time (Java compiler)

JNI Java Native Interface

JVM Java Virtual Machine

KB kilobyte (1,024 bytes)

LCU logical control unit

LOB large object

LPAR Logical Partition

LPL logical page list

LRECL logical record length

LRSN log record sequence number

LVM logical volume manager

MB megabyte (1,048,576 bytes)

MSM Multidimensional Storage Manager

NPI non partitioning index

NVS Non Volatile Storage

ODB object descriptor in DBD

ODBC Open Data Base Connectivity

OLAP Online Analytical Processing

OS/390 Operating System/390

PAV Parallel Access Volume

PDS partitioned data set

PIB parallel index build

PSID page set identifier

PSP preventive service planning

PTF program temporary fix

PUNC possibly uncommitted

QBIC query by image content

QMF Query Management Facility

RACF Resource Access Control Facility

RBA relative byte address

RECFM record format

RID record identifier

ROT rule of thumb

RR repeatable read

RRS resource recovery services

RRSAF resource recovery services attach facility

RS read stability

RSM Relational Resource Manager

RTS real time statistics

RVA RAMAC Virtual Array

SDK software developers kit

SMIT System Management Interface Tool

SVL IBM Silicon Valley Laboratory

TCB Task control block

USS UNIX System Services

WAS WebSphere Application Service

WLM Workload Manager

WSF worksheet file format

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 334.

- *DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite*, SG24-6289
- *DB2 for z/OS and OS/390 Version 7 Performance Topics*, SG24-6129
- *DB2 for z/OS and OS/390 Tools for Performance Management,* SG24-6508
- *DB2 UDB Server for OS/390 and z/OS Version 7 Presentation Guide*, SG24-6121
- *DB2 for z/OS DM Tools for Database Administration and Change Management*, SG24-6420-00
- *DB2 Web Query Tool Version 1.2*, SG24-6832
- *A Practical Guide to DB2 UDB Data Replication V8,* SG24-6828
- *DB2 Warehouse Management: High Availability and Problem Determination Guide*, SG24-6544
- *Large Objects with DB2 for z/OS and OS/390,* SG24-6571

Other resources

These IBM publications are also relevant as further information sources:

- *DB2 UDB for OS/390 and z/OS Version 7 Utility Guide and Reference,* SC26-9945-03
- *DB2 UDB for OS/390 and z/OS Version 7 Application Programming and SQL Guide,* SC26-9933
- *DB2 UDB for OS/390 and z/OS Version 7 Programming Guide and Reference for Java,* SC26-9932
- *DB2 UDB for OS/390 and z/OS Version 7 Administration Guide,* SC26-9931-03
- *DB2 UDB Quick Beginnings for DB2 Servers Version 8,* SC09-4836
- *DB2 UDB Administration Guide: Performance Version 8*, SC09-4821
- *DB2 UDB Administrative API Reference*, SC09-4824
- *DB2 UDB Data Movement Utilities Guide and Reference Version 8,* SC09-4830
- *DB2 UDB Federated Systems Guide Version 8,* SC09-4830
- *DB2 UDB Command Reference Version 8,* SC09-4828
- *DB2 Connect User’s Guide Version 8*, SC09-4835
- *DB2 High Performance Unload for z/OS Version 2 Release 1*, SC27-1602
- *DB2 High Performance Unload for Multiplatforms Version 2 Release 1*, SC27-1623-01
- *DB2 Data Export Facility for z/OS Version 1 Release 1,* SC27-1466
- *DB2 Replication Guide and Reference*, SC26-9920-00

- *DB2 Web Query Tool User's Guide*, SC27-0971-05
- *Data Warehouse Center Administration Guide*, SC26-9993-01
- *Information Integration, IBM Systems Journal - Vol. 41, No. 4, 2002,* G321-0147

Referenced Web sites

These Web sites are also relevant as further information sources:

- DB2 UDB distributed V8 beta code and manuals

  http://www.ibm.com/db2/v8beta/
- IBM Data Management home page

  http://www.ibm.com/software/data
- IBM Data Management Tools

  http://www.ibm.com/software/data/db2imstools/
- DB2 support, online manuals

  http://www.ibm.com/software/data/support
- DB2 service and FixPaks

  http://www.ibm.com/cgibin/db2www/data/db2/udb/winos2unix/support/help.d2w/report
- DB2 white papers

  http://www.ibm.com/software/data/pubs/papers
- DB2 for UNIX and Windows manuals and support

  http://www.ibm.com/software/data/db2/udb/winos2unix/support
- DB2 UDB for Windows

  http://www.ibm.com/software/data/db2/udb/udb-nt
- DB2 UDB LUW manuals

  http://www.ibm.com/cgi-bin/
- DM Tools support information

  http://www.ibm.com/support/search
- DB2 for z/OS Version 8 information

  http://www.ibm.com/software/data/db2/os390/db2zosv8.html

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

F

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Moving Data Across the DB2 Family

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Moving Data Across the DB2 Family

Explore the usability and performance of the DB2 Family's functions

Try out the High Performance Unload on several platforms

Experience the new Load from Cursor utility option

Moving data across different databases and even different platforms has been a common task in IT shops for quite some time. Applications may have been developed independently and over time, using packages and different technology; and data might reside on different platforms exploiting the specific platform strong points. However, there still is a growing need for applications that need to access all of this data for overall processing.

While new Web related technologies are emerging with the intent to provide functions to collect and integrate information across multiple databases and applications for access in real time, moving data to one location for overall processing is still a very common requirement.

This IBM Redbook provides an overview of what is currently available within the DB2 Family of products (specifically DB2 for z/OS, and DB2 for UNIX and Windows) in terms of functions, tools, and utilities to satisfy the need for moving data. We focus on discussing High Performance Unload and Cross Loader; the first one is a tool, the second one is a new option of the Load utility, since they are the latest functions that IBM has released. We also introduce the concepts and some examples of using the Federated Database support.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6905-00 ISBN 0738428299