IBM

DB2 UDB V7.1
Porting Guide

- A practical guide to porting Sybase Adaptive Server to DB2 UDB
- Techniques and considerations for migration projects
- Databases, applications, and data conversion

Tetsuya Shirai
Kenji Horino
John Russell
John V. Partridge Jr.
Art Sammartino

Redbooks

ibm.com/redbooks

SG24-6128-00

International Technical Support Organization

DB2 UDB V7.1 Porting Guide

December 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix C, “Special notices” on page 247.

First Edition (December 2000)

This edition applies to Version 7, Release 1 of IBM DB2 Universal Database, Program Number 5648-D48 for use with the AIX V4.3.3 Operating system.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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

Contents

Figures

Tables

Preface

This IBM Redbook is intended to help database administrators and system designers perform database and application conversion from Sybase to DB2 Universal Database (DB2 UDB) Version 7.1. It contains a description of the conversion process and suggestions on how the mapping of database features may be accomplished.

The target audience of this redbook is database administrators and system designers with Sybase and/or DB2 UDB for AIX background.

This book was written from the AIX operating system perspective, and some commands shown in this book may not be available on other operating systems. However, all sections in this redbook except some operating system specific commands and operations should be applicable for database systems on non-AIX operating systems such as Solaris and Windows NT.

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.

Tetsuya Shirai is a Project Leader at the International Technical Support Organization, San Jose Center. He worked in Tokyo, Japan before joining the San Jose ITSO. He has been with IBM for eight years, working for the last four years with DB2 UDB products. He has worked in the area of DB2 services, teaching classes to both customers and IBMers. He has also provided consulting services to DB2 customers worldwide.

Kenji Horino is an IT Specialist in Japan. He has six years of experience in DB2 Universal Database and provides support for customers using DB2 UDB on the PC and UNIX platforms. His areas of expertise include designing and performance tuning for data management systems. He worked as a DB2 DBA in the Nagano Olympic project and achieved great success.

John Russell is a Sybase DBA at a customer's location for IBM Global Services. He has been with IBM Global Services for nine years and has a total of 33+ years experience in the IT profession serving in a variety of technical and management roles prior to joining IBM as part of an outsourcing agreement in 1991. He has also done over 30 Operating System upgrades and conversions for customers in the DOS/VSE arena. He has been a Sybase DBA for five years, and has had considerable experience with

Sybase releases 4.92, System 10, System 11, and Sybase Adaptive Server release 12.0, as well as some Oracle and DB2 experience.

John V. Partridge Jr. is an IT Specialist with the Software Migration Project Office (SMPO). He has eight years of experience in database management systems including DB2 and Sybase, six years as an independent database consultant, and four years in designing and implementing data warehouse solutions.

Art Sammartino is a Software IT Specialist with the Software Migration Project Office (SMPO). He has been working for IBM for one year providing technical support, and engaging in proofs-of-concept, for competitive relational DBMS migrations to IBM DB2. Art is based in White Plains, NY.

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

Nick Samanic
Marina Greenstein
Dominic Marrese
Martin Spratt
IBM Software Migration Project Office

Patrick Dantressangle
IBM Silicon Valley Laboratory

Amyris Rada
IBM Toronto Laboratory

John Field
IBM Watson Research Center

Paolo Bruni
Mary Comianos
Emma Jacobs
Yvonne Lyon
Deanna Polm
International Technical Support Organization, San Jose Center

Comments welcome

Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks review” on page 263 to the fax number shown on the form.
- Use the online evaluation form found at **ibm.com**/redbooks
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction

This IBM Redbook is intended to help database administrators and system designers who perform database and application porting from Sybase Adaptive Server to DB2 Universal Database (DB2 UDB) Version 7.1. We have chosen a sample scenario based on a subset of a real customer's application running on Sybase Adaptive Server and we have used it throughout the project. In this book, we will describe the porting process which we have performed and give suggestions on how the mapping of database features can be accomplished.

We have used ManTech SQL Conversion Workbench (SQL-CW) Version 3.0 Beta, which is also called the Stored Procedure Converter Tool (SProCT), throughout the project. However, since the migration tool technology is changing and you may want to use different tools, or even perform migration projects without tools, we avoid showing just how to use SQL-CW in each chapter. Rather, we explain the most important aspects of porting databases and applications, including differences in database options, data definition languages (DDL), SQL statements, data conversion, and application conversion. Appendix A, "Conversion tools" on page 229 contains a brief description of SQL-CW. See this appendix for ideas on how to use SQL-CW.

The application we have chosen cannot be valid for all possible permutations of application variables. Therefore, we have also tried to go beyond our sample scenario and to extrapolate to more general considerations applicable to a larger variety of environments.

This chapter contains an introduction to the project and a summary of our conclusions.

1.1 Decision to port databases and applications

Porting databases and applications across different database management systems (DBMSs) is certainly not a trivial task. The decision to port is generally made at a high level, at a time when there is full justification in terms of costs and expected returns on investment. The major issues that bring up the need to port, and which are the main components for building a business case, are related to the following areas.

- Performance:

  Aspects of performance include scalability, availability, data movement, response time, and the ability to support multiple query workloads.

- Configuration costs

  Realistic cost assessment is required, based on overall development, maintenance, and tuning cost and the full exploitation of current investments, both in terms of skill sets and reduced licence costs.

- Data infrastructure

  Data is no longer an application-specific resource, but an enterprise-wide tool to provide critical business information for a competitive advantage. Often it is not enough to navigate through the data, but it is necessary to invest in the infrastructure in order to more easily integrate enterprise views of data.

  The deployment of enterprise resource planning, customer relationship management, and business intelligence systems may raise the need to reconsider the current choice of environment for important applications as well as to appreciate the strength and value of DB2 UDB.

DB2 UDB is a database leader in several technologies, and offers true multi-platform support, reliability, and scalability from a single processor to symmetric multiple processors, and to massive parallel clusters with terabytes of data and/or thousands of users. This can be a major motivation to convert your database system to DB2 UDB.

Also, a large number of industry solution providers have adopted DB2 UDB to support their applications or adapted their tools to support DB2 UDB. These solution providers include SAP, Baan, Peoplesoft, Siebel, and more. You can buy these solutions ready-made to meet your requirements, rather than developing them by yourself. This is another reason to choose DB2 UDB.

1.2 Project overview

The project consisted of taking an existing customer application based on Sybase Adaptive Server and porting a meaningful subset of it to use DB2 UDB for AIX.

The chosen application is a stored procedure based application which is a centralized scheduling tool for administering DBA type functions such as updating statistics and dumping databases and logs for multiple DBMS clients (ORACLE, Sybase, Informix, MS SQL Server, and so on). Please see Chapter 2, “Project scenario” on page 5 for more information on the customer application chosen.

The database design, the data, and a subset of the application were successfully converted, although unfortunately, in the short time available, we were unable to port all the application codes, including 115 stored procedures. Nevertheless, we gained enough experience to share with you and demonstrate the viability of the porting from Sybase Adaptive Server to DB2 UDB.

1.3 Summary of considerations

In our project, we successfully converted the database design, the data, and a subset of the application from Sybase Adaptive Server to DB2 UDB Version 7.1.

Both Sybase and DB2 UDB are based on the relational logical model, and we found that they had many similarities in terms of functions. Thus, converting the database objects including the tables and indexes from Sybase to DB2 UDB was not such a challenging task.

Moving the actual raw data from Sybase to DB2 UDB was not a complicated undertaking either. We unloaded the data from the source Sybase tables to delimited ascii files using the Sybase BCP utility and populated the target DB2 UDB tables using the DB2 IMPORT or LOAD utility. In our scenario the amounts of data were small. If large amounts of data must be moved, using the LOAD utility would certainly be the more efficient approach, providing better performance and allowing no logging.

For `DATETIME` and `SMALLDATETIME` columns of the Sybase source tables, we had to use `SELECT` statements with the `CONVERT` function to unload and change the format of the data. This was because the Sybase BCP utility unloads `DATETIME` or `SMALLDATETIME` data into the format which is not acceptable to the DB2 IMPORT or LOAD utility.

To convert the data, we experimented with an alternative solution using DataJoiner. DataJoiner enabled us to unload data from Sybase tables into integrated exchange format (IXF) files which also contain table and index definitions. Using DataJoiner made the database and data conversion process easy because we could accomplish the tables, indexes, and data conversion at a time by executing the IMPORT utility with these unloaded IXF files.

Porting the stored procedures was a more complicated but certainly successful task. The factor that contributed to complexity was Sybase Transact-SQL, which has its own extensions in addition to the SQL92 at the Entry-level, and was used all the stored procedures in our scenario. We

converted Sybase SQL stored procedures into DB2 SQL stored procedures, which is available in DB2 UDB Version 7.1, and confirmed that the stored procedure conversion from Sybase to DB2 was feasible.

Porting triggers was also a challenging task. Since Sybase triggers allow a wider variety of SQL statements to be written in the trigger body than DB2 triggers do, we needed to make some modifications in the trigger body. We will show how we could convert Sybase triggers in 7.7, “Trigger conversion” on page 155.

We used the ManTech SQL Conversion Workbench (SQL-CW) Version 3.0 Beta throughout the project and it was the great contributing factor to our success. It generated all the DDLs to build the database objects in the target DB2 database, script files which could unload and load data with small modifications, and enabled us to perform the automatic conversion of the stored procedures. We strongly recommend using such a conversion tool to assist you when migrating from Sybase to DB2 UDB. Please check the following Web site for the latest information on the migration tool from Sybase to DB2 UDB:

```
http://www-4.ibm.com/software/data/db2/migration/
```

The feasibility of the porting operation was proved; it can certainly be done, given circumstances similar to the pilot effort, and assuming that there are sufficiently experienced and skilled people available on both sides of the conversion so that the objectives can reached quickly. Also, communication with the site systems personnel needs to be clear, effective, and flow freely, since system support was crucial to the success of our project.

Chapter 2. Project scenario

This chapter describes our project scenario, which consisted of a customer's application running on the Sybase Adaptive Server.

2.1 Source database system

The chosen customer application is a stored procedure base application, which is a centralized scheduling tool for administering DBA type functions such as updating statistics and dumping databases/logs for multiple DBMS clients (ORACLE, Sybase, Informix, MS SQL Server, and so on).

This application is initiated via the *cron*, which is a UNIX daemon process that can invoke shell commands at specified dates and times. The application wakes up every minute to check a schedule table in the Sybase database and determine whether any task is scheduled to run. If there is a task which needs to run, then the application executes a script processing a specific activity, such as updating statistics, dumping databases, and so on.

Figure 1 shows an overview of the application.

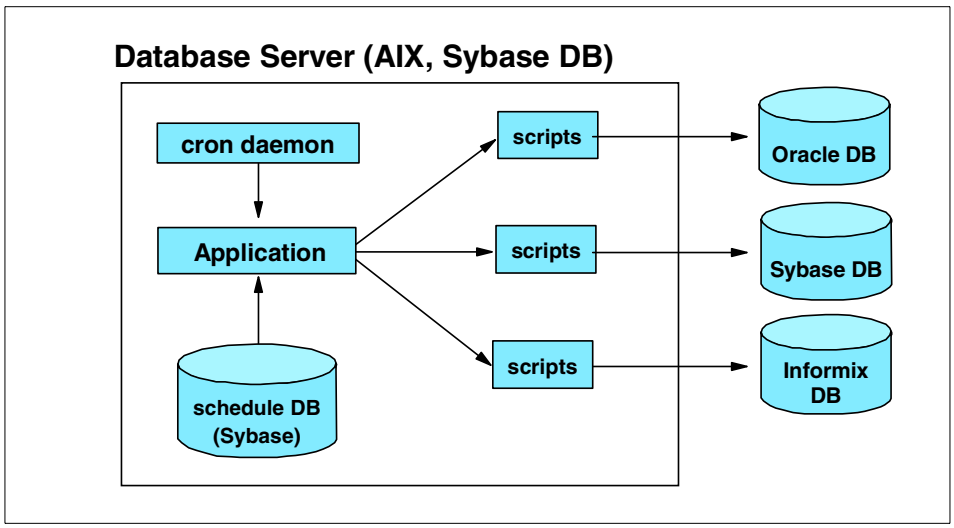

Figure 1. Application overview

2.2 Project scope

The source Sybase database consists of 35 tables, 30 views, 30 indexes, 115 stored procedures, and 16 triggers.

As mentioned in 1.2, “Project overview” on page 2, not all of the database objects (including 115 stored procedures) have been ported to DB2 UDB in our project. Our goal was converting all the database design, the data, and a subset of the application to confirm the feasibility of the conversion from Sybase to DB2 UDB, and to find common pitfalls and workarounds.

Figure 2 shows our lab environment. We have installed both Sybase Adaptive Server and DB2 UDB on the same RS/6000, and we perform the database conversion, data conversion, and application conversion within the same machine.

Throughout the project we utilized the ManTech SQL Conversion Workbench (SQL-CW), which works on the Windows NT workstation. SQL-CW extracts the metadata describing the source database structure and generates DDLs and unload/load scripts to perform the database and data conversion from Sybase to DB2 UDB. SQL-CW generates all the DDLs and scripts on the local Windows workstation, therefore, we have executed these DDLs and unload/load scripts from the Windows workstation though the Sybase Client and DB2 Client.

As mentioned in 1.2, “Project overview” on page 2, in addition to the data conversion using the unload/load scripts that SQL-CW generated, we experimented with an alternative solution using DataJoiner.

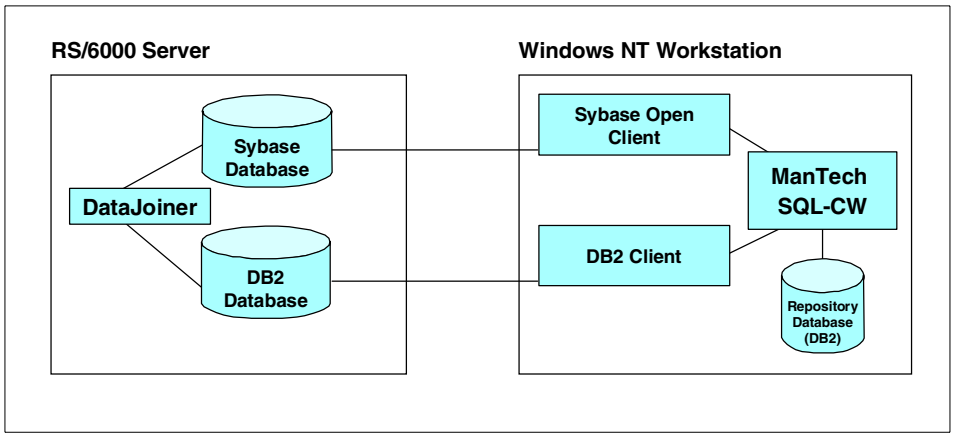

Figure 2. Lab environment

2.3 Hardware and software configuration

This section describes the hardware and software configuration used for the lab environment on which our project was carried out.

Hardware configuration

As shown in Figure 2, our lab environment consisted of an RS/6000 server and Windows NT clients. Their specifications were as follows:

- RS/6000 J50 (4 x 200 MHz CPUs with 1 GB of RAM, and 22.5 GB of disk)
- IBM PC 300 GL (333 MHz CPU with 256 MB of RAM, and 15 GB of disk)

Software configuration

On the RS/6000 server, we have installed the following software:

- AIX 4.3.3 + PTF
- Sybase Adaptive Server for AIX Version 12
- DB2 Universal Database for AIX Enterprise Edition Version 7.1 + FixPak 1
- IBM C/C++ for AIX Version 3.6.6
- DataJoiner for AIX Version 2.1.1

On the PC client, we have installed the following software:

- Sybase Open Client Version 12
- DB2 Universal Database for Windows NT Enterprise Edition Version 7.1+ FixPak 1
- ManTech SQL Conversion Workbench (SQL-CW) Version 3.0 Beta

Chapter 3. Conversion process

This chapter looks at the conversion process and the different steps involved in the planning and execution of a successful conversion. For further information, refer to *Planning for Conversion to the DB2 Family: Methodology and Practice*, GG24-4445.

The topics covered are:

- Strategies available
- Conversion methodologies
- Planning the conversion
- Conversion considerations

3.1 Strategy and conversion methodologies

In performing a conversion, you need to decide on the following:

1. The overall strategy

   This is the general approach to the question of moving to DB2 UDB. Is speed the most important parameter, or perhaps the method involving the least amount of risk?

2. The starting point

   What is the first application to be moved to DB2 UDB? Is it just part of a total application? How will coexistence be handled?

3.1.1 Strategy definition

There are several strategies that could be used to convert the system, each may be better suited to different environments. Table 1 on page 11 summarizes the strategies available and lists the advantages and disadvantages of each.

3.1.1.1 Big Bang

In this strategy, all applications and data are moved to DB2 UDB and go live simultaneously. This is useful when speed of conversion is the overriding consideration; however, this strategy has the highest risk.

3.1.1.2 Piece by Piece

In this strategy, definable applications or pieces of them, and the data that they use, are migrated to DB2 UDB one at a time. This strategy implies that all applications and data will eventually be converted to DB2 UDB. No enhancements are made to the application during its conversion, and future enhancements are made in DB2 UDB. This strategy is indicated when the primary considerations are to reduce risk, keep track of projects, and gain experience as the work proceeds.

3.1.1.3 Tight Coexistence

This strategy means that the old systems are kept and new systems are developed with new enhancements for using data in DB2 UDB. As systems come up for redevelopment, they are redeveloped using DB2 UDB.

3.1.1.4 Loose Coexistence

This strategy usually involves a management information system (MIS) or reporting system. A copy of the data is loaded into DB2 UDB and accessed using report programs. This strategy means some of the data is kept multiple times, but operations continue as normal. Copies of the data can be used as part of conversion, and they can be kept up-to-date by asynchronous update or periodic copy.

3.1.1.5 Combinations

A combination of strategies may be the best solution. One option is to set up a read-only MIS system, first to handle reporting, convert a few existing report programs to use the MIS and provide extra reporting through DB2 UDB.

Where the Big Bang strategy is not appropriate, it may be necessary to migrate some applications piece by piece and then follow up with a series of “little bangs” where two or three applications are migrated together with their common data. It may be that some applications are in good shape, but others are in poor shape and need rewriting. This may mean that different methods will be employed for different applications, but it does not alter the basic strategies.

Table 1 summarizes the main advantages and disadvantages of each strategy.

Table 1. Summary of strategy characteristics

Strategy	Advantages	Disadvantages
Big Bang	Fastest for total conversion	High risk
	No coexistence problems	Long time before first benefits
	No duplicated work force	Complex change management
Piece by Piece	Low risk	Long project
	Fast way to get some benefits of DB2 UDB	Need to handle coexistence
	Workforce adjusts overtime	Dual platforms for some time
Tight Coexistence	Enhancements added	Need very good dual update mechanism
	Tools possible	Need to open programs on multiple occasions
	Each conversion separately justified	Dual platforms indefinitely
Loose Coexistence	Management information greatly enhanced	Does not fix operational problems
	Low risk	Hard to get up-to-the-minute information
	Few coexistence problems	

3.1.2 Conversion methodologies

There are several methodologies that could be considered, any of which could be correct in different circumstances. Once a conversion methodology has been selected, it will then set the way for data design: programs, testing, tools, and any data cleaning.

Table 2 on page 14 gives a summary of the conversion methodologies available and their advantages and disadvantages.

3.1.2.1 Translation

Translation occurs when the application Database Management System (DBMS) calls are translated one-for-one into DB2 SQL calls. When applied to data, translation means that the data is also copied one-for-one into DB2 UDB with no allowances made for the advantage of the large number of DB2 data types.

Translation means modifying the programs to access the new database and using the old data layout for the data; it is a one-for-one line translation of the data access language in the source code. Translation tries to make no changes to the business or data logic parts of the programs and no structural changes to the data layout. Where speed of conversion is the highest priority, translation allows for fast migration.

In our project, we have used the translation conversion method.

3.1.2.2 Transparency

With transparency, the data is migrated to DB2 UDB, and a special program is written to intercept calls to the old DBMS. This program will translate the calls into DB2 SQL transparently and routes the call to DB2. The program remains unchanged and can be executed with the data in either database system. Transparency enables all the data to be moved to DB2 and the programs to be rewritten or changed one at a time. It also offers a migration path for businesses with very large databases that require very high availability and administrators who do not have time to unload and reload the data.

DataJoiner is a product from IBM that can help with this method. It allows transparent access to data wherever it may reside with the application, while being unaware of where the data is coming from. By using DataJoiner, applications can access the DB2 and Sybase databases at the same time, thus allowing you to join a DB2 table with a Sybase table in one SQL statement. DataJoiner can also be used to assist in some of the other methodologies which use a coexistence strategy. It can also be used as a tool to move the data from Sybase into DB2 UDB (see 6.2, “Data conversion using DataJoiner” on page 104).

3.1.2.3 Re-engineering

Re-engineering allows us to alter the data and programs to be compatible with good DB2 UDB design without having to rethink the whole design.

The data is adapted to a new data model, and the program data calls are adapted to fit the new target data model. For the applications, only the database calls and data logic parts of the program are changed with limited alterations to the database logic consistent with the new design. Re-engineering would have only minimal impact on the business-logic part of the program.

This method is good for testing for equivalent function and for future enhancements. It should perform well, but will take longer than translation.

3.1.2.4 Reverse engineering

Reverse engineering is a form of re-engineering that involves capturing the old design into models, modifying them, and generating new programs and a new DBMS design. Reverse engineering usually means using tools to recover the design of the original data into a reconstructed data model and recover the application into a new process model. These models can then be modified to improve the design. The new improved models are then used to generate (forward-engineer) DB2 Data Definition Language (DDL) for the database and new code for the applications.

Reverse engineering is a very valid way to improve applications and data, but to be really effective, good tools are needed. Tools such as PLATINUM Erwin are available to help automate this process.

3.1.2.5 Redevelopment

Here, the whole application is redeveloped according to user requirements. In the case of data, it means that the data model is rethought from scratch using requirements from the users. Redevelopment can use the data modeling tools. If data modeling tools are used to model the process, future enhancements mean only changing the model and regenerating code.

Choosing this course of action requires the purchase of a suitable package since this is equivalent to writing an application suite from scratch.

3.1.2.6 Corporate model

This refers to the generation of corporate-wide models. Instead of just one application suite, the entire business can be modelled into a corporate business data model and a corporate business process model. The DBMS design and code may be generated from the models and any changes or enhancements needed are reflected in the model's original and newly generated code. Each application suite may be rewritten or regenerated one by one to access a new corporate-wide database.

Although costly in time and effort, it is a method that potentially provides the greatest benefits. It allows systems to be brought together using an integrated corporate database.

3.1.2.7 Combinations

To some extent, the methods can be mixed. It is possible to use one method for data and a different method for the applications. There is the possibility of using one method and switching to another. It may be that time is of the essence and therefore a quick translation just to move the DBMS is required.

As soon as that conversion is complete, sections of the DBMS and relevant programs can be re-engineered.

Table 2 summarizes the main advantages and disadvantages of each conversion methodology.

Table 2. Summary of conversion methodologies

Methodology	Advantages	Disadvantages
Translation	Easiest method	Few advantages of DB2 UDB
	Easy for tools	Design inflexible for future
	Possible to use in two-stage approach	
Transparency	Data may be restructured	Performance maybe a problem
	Applications can be rewritten later	Module difficult to write
	Low risk	Conversion takes much longer
Re-engineering	Obtain advantages of DB2 UDB	Relatively longer time
	Allows limited redesign	
Reverse engineering	Design may be optimized for performance	Tools do not handle all situations
	Later enhancements easier	Less efficient code from tools
Redevelopment	Known process	Much longer development time
	Future maintenance easier	Existing investment lost
Corporate models	Business modeled as a whole	Large up front investment
	Less redundant code	Existing investment lost
	Package tools available	Need to manage changes

3.2 Planning the conversion

A DBMS conversion can sound simple — just collect all the programs and change the DBMS calls from the old data language to DB2 SQL calls. However, consider these questions:

- Which application will be converted first?
- What is known about the makeup of the programs?
  - How many are there?
  - Are they all the same language?
  - Do they all make DBMS calls?
  - How good is the current documentation?
  - Do all programs need converting?
- Are the applications independent?
- If the data is moved to DB2 UDB for use by the new system, how will updates be synchronized?
- Can a single application system be split so that it can be moved a piece at a time?
- How much time is allocated for the conversion process?
- How good is the data model of the existing system?
- How will the results be tested?

These are just some of the considerations that must be fully addressed to have a successful conversion. A decision must also be made as to which strategy and conversion methodologies to use.

These issues are addressed via the Three Stage Approach, as shown in Figure 3.

1. Stage one

   Before any conversion can be contemplated, there is a need to take stock of the current systems. This stage takes an overview (portfolio analysis) of the system and considers what options will fulfill most requirements to produce a cost effective conversion strategy.

2. Stage two

   The second stage tests the feasibility of the output produces by stage one. It puts plans in place and tests them with a pilot conversion to see if the assumptions made are correct.

3. Stage three

Stage three is the main conversion. It is where the majority of the work takes place. This is implementing the plan worked out in Stage Two, then cutting the new system over to production.

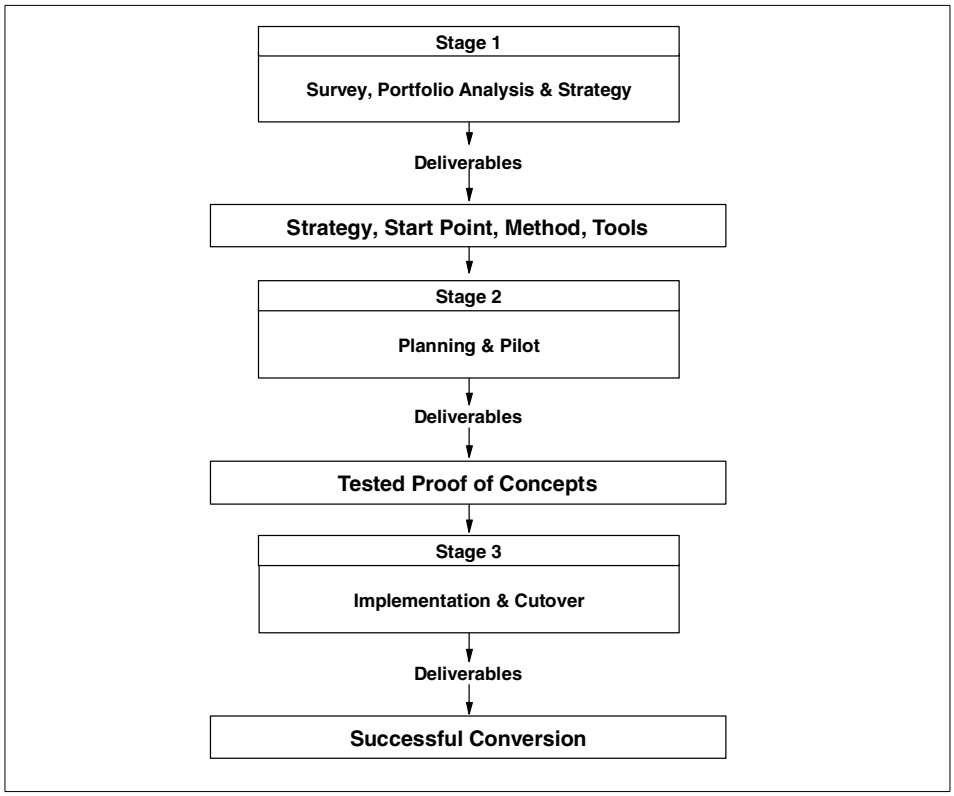

Figure 3. Three stages of conversion

3.2.1 Stage one: defining the strategy

Before any conversion can be contemplated, current systems need to be evaluated. The information obtained in stage one is essential for deciding which strategy and which conversion method to use.

3.2.1.1 Survey

This is a short task where the area of the proposed conversion is set down on one page. The purpose of this short document is to set the boundaries of the subsequent project. It sets out which system or systems are candidates for the conversion, the programming languages involved and the data sources.

3.2.1.2 Business requirements

The most important aspect of conversion is contained here. This sets out what is expected from the conversion and is used to guide later decisions. It effectively sets down the business requirements and other criteria that must be achieved for the project to be a success.

3.2.1.3 Portfolio analysis

The goal of the portfolio analysis is to get an idea of the state of the systems, the size of the task, the effects on data used by multiple systems and any special difficulties to account for.

3.2.1.4 Strategy definition

There are several ways that you could go forward and many methods you could use to actually convert the system, all of which could be correct in some instances.

Now that the present position is understood, the main decisions can be made about the applications. These include:

- Whether to adopt a strategy to move everything at once (Big Bang) or to move small pieces at a time (Piece by Piece).
- Whether to move all the reporting programs
- Which application to move first
- Which programs and data to use for a pilot

3.2.1.5 Conversion methods

With the overall strategy set, the method can then be selected and the requirements set for the data layout and DB2 features, including:

- What method of conversion will suit the programs and data best
- What to do about data cleaning
- What testing strategy to use
- What enhancements should be made, and when

3.2.1.6 Deliverables

The deliverables of this first stage are:

- A description of the business benefits expected
- An understanding of the migration process
- An overall strategy
- The place to start the migration process

- Scope of the pilot
- List of issues

The biggest factor influencing success is going into the new territory with awareness of both the pitfalls and the rewards. Working through Stage One will give an understanding of the process, a set of requirements, and the first gross estimate of the conversion task.

3.2.2 Stage two: testing the concept

The objective of stage two are to put proper plans in place and to test the theories with a pilot conversion, the results of which are then used to adjust the plans for the final stage. There are a number of relatively small, but vital, tasks that need to be done:

Data layout The database design must be carefully checked and a first cut DB2 design produced.

Programs The programs need checking too. An inventory of where they are by categorizing them into Online Transaction Processing (OLTP), batch and reporting applications.

Testing The test strategy decided in stage one needs to be written, set up and tested.

Performance A performance plan needs to be established. It should identify critical processes and specify how changes are to be made.

Change control A change control system must be put in to place.

Data movement A plan needs to be put in place for data movement, that is, the unloading, reformatting and loading of the actual data.

Set up DB2 UDB If DB2 UDB is not installed, then installation needs to take place before any pilot can run. If DB2 UDB is already in use, then it is preferable to establish a separate system for the conversion work.

Pilot The pilot takes the previously chosen applications and runs them through the conversion process.

Review At the end of the pilot, a thorough review should be made, and the results should be fed back into the plans and adjustments made.

3.2.2.1 Deliverables

The deliverables for stage two can really be expressed as a tested proof of concept. Specifically, there should be:

- Tested proof that all the components work
- A program inventory
- A data inventory
- Old data column to new data column cross reference
- Old program name to new program name cross reference (if applicable)
- A project plan for the entire conversion:
  - Application plan showing how the changes will be tackled
  - Data movement plan
  - Performance test plan
  - Change control plan
  - Data cleaning
  - Test plan
- Database design
- Critical process list
- A data movement plan
- Documented results of the pilot
- The first few converted programs
- Results of the review

Done properly, the second stage shows that the project is viable, highlights any areas of weakness that can be corrected and provides feedback to be used in the main and final stages of the plan.

3.2.3 Stage three: implementation and cut-over

By this time, although there is much left to do, the rest should be straightforward. It is now a matter of making sure the plan is correctly implemented.

3.2.3.1 Implementation

This is the implementation of the plans worked out in stage two. Here the programs are converted, tested and changed, if required, via the change control plan already set up.

3.2.3.2 Data cleaning

The accuracy of the actual data needs to be assessed for quality. Depending on the errors, this may require anything from a small to a significant investment. The most common corruption is that data is entered incorrectly in the first place. Sometimes data has been entered incompletely, and the additional data was never entered.

During the data load using the DB2 LOAD/IMPORT utility, the utility checks the properties of the data, and it will refuse to load any records that are obviously wrong, placing them instead in a discard file. This is one method that can be used for assessing errors; however, the utility cannot detect where the wrong values have been entered.

The data cleaning strategy identified in stage one and tested in stage two should be used to verify the data and correct any errors.

3.2.3.3 Data cut-over

Here the data is unloaded, reformatted and loaded into DB2. The plans for the applications are bound, the libraries switched and the new system brought up.

3.2.3.4 Testing

Testing is very likely to take 50 percent or more of the time taken to convert the DBMS. Testing will be an iterative process and it is important that problems are documented and the change control process is followed. During conversion, testing differs from normal application development in these ways;

1. There is a need to test every function and every part of the code to ensure that it all still performs in the same way.
2. The tests must be repeatable and repeated to ensure that errors are not reintroduced.
3. The tests must ensure that the program still functions the same.
4. Automated test tools can shorten the time taken for testing by providing repeatable automatic comparison and regression testing.

5. Stress testing needs to be a part of the testing to ensure that the whole application will function well and to check that things, such as deadlocks, have not been introduced.

3.2.3.5 Fallback plan

In case of unforeseen problems, a fallback plan needs to be set up. This should be business-as-usual, as customers are used to changing levels of software, both from themselves and from software vendors.

3.2.3.6 Production cut-over

The production cut-over process should be carefully planned after ensuring all applications are working properly and performance issues are resolved. Care must be taken to coordinate the recreation of the data, if necessary to facilitate a clean cut-over.

3.2.3.7 Consolidation

The period of post-live consolidation needs to be defined. The system needs to run for a month or so before any new enhancements are added. Further, when the conversion is declared complete, do not delete the old system; archive it.

3.2.3.8 Deliverables

This is, of course, the new system working as planned with DB2 UDB.

3.3 Conversion considerations

This section highlights some of the considerations found during the many conversions to DB2 that have already been completed. These are not unique to a particular environment and may very depending on your specific environment; therefore any, all or none of these considerations may apply.

- One percent of source code is lost.

  When it comes to actually converting every program, some source code cannot be found.

- Ten percent of the source code does not match the production object code.

- Sixty percent of the program inventory needs to be converted.

  This relates to redundant programs, unused programs, reporting programs, programs without any DBMS calls, and those that need rewriting anyway.

- Testing and fixing will take over 50 percent, and may take up to 80 percent of the project time.
- Sixty percent of program problems are unrelated to conversion.

  Problems due to unsuitable conversion do occur, but the majority of issues that surface are due to:

  - Bad source code
  - Known bug in original code
  - Latent bug that is found, due to new environment
  - Latent bug that is found, due to more comprehensive testing

Chapter 4. Database structure and data types

This chapter discusses and compares the database structures and supported data types between Sybase and DB2 UDB.

4.1 Database structure comparisons

The Sybase environment consists of the Sybase database server and one or more user databases within the Sybase database server. There can be multiple database servers to support the user's requirements. There will always be a separate server process for the backup functions.

DB2 uses instances to support the databases within the machine. A DB2 instance can manage multiple databases. When a database is created, the database manager creates a separate file system subdirectory for the control files and table spaces. Multiple instances can be used also to support user requirements. The backup utility for DB2 UDB does not require a separate instance.

4.1.1 Sybase database structure

The Sybase server physical structure consists of databases and UNIX files. Databases are used for Sybase system and user data. UNIX files are used for the configuration file and the interfaces file as well as server startup scripts.

4.1.1.1 UNIX files

The following UNIX files are used:

- The configuration file
- The interfaces file
- The server RUN files

When you create the Sybase server using the `sybinit` command, you will create the configuration file and the interfaces file in the UNIX directory you specified in your `SYBASE` environment variable.

The configuration file contains the options for the Sybase server. The naming convention is `server_name.cfg`. You may update the configuration file by using the `sp_configure` stored procedure within the database, or you can edit the file. If you use the `sp_configure`, the server renames the current file to a backup file in the format of `server_name.nnn` where `nnn` is a simply the next sequential number. If you update the file using a text editor such as `vi`, it is up to you to save a backup of the file.

The interfaces file is used by the Sybase server to communicate with the Open Client. It contains, among other things, the communications protocol type, the port address, retry counts.The interfaces file can be updated using the `sybinit` utility or a text editor such as `vi`. If you use a text edit, you must insure that each line begins with the tab character. The `sybinit` utility also preserves a backup copy of the file.

The server RUN files are UNIX shell scripts created by the `sybinit` utility when you installed the Sybase server. The file will be created in the `install` subdirectory of the path you entered for the `SYBASE` environment variable. The naming convention is `RUN_`*`server`* where *`server`* is the name you specified in the install process. There will be a RUN file for each server installed, including the Backup server. These files can be edited using a text editor if you need to add or change startup options.

4.1.1.2 Databases

The Sybase server will have the following databases:

- The `master` database
- The `sybsystemprocs` database
- The `sybsystemdb` database (optional, used for two-phase commit)
- The `model` database
- The `tempdb` database
- The `dbccdb` (optional) database
- User databases (as required)

The `master` database contains global system tables for use by the Sybase server, for example, the `sysdevices` table contains information about each device assigned to the server.

The `sybsystemprocs` database contains the Sybase suppiled system stored procedures.

The `sybsystemdb` database contains the table `spt_committab` which is used if you are using the two-phase transaction commit feature.

The `model` database is used to provide the template for database creation. You may specify database options such as ‘select into/bulkcopy’ that you want as a default for any database created using the create database command.

The `tempdb` database is used for temporary storage as needed, for example temporary tables.

The dbccdb database is an optional database you can set up to store options and output data from the Sybase database consistency check (dbcc) utility.

The user databases are those you create to contain your application data.

4.1.1.3 Sybase data management

Sybase data management uses devices, segments, and tables.

Sybase devices

Devices are defined to Sybase through the disk init command, and size is determined at initialization time. Once initialized, the size cannot be changed, neither decreased or increased. If additional space is required for the database, more devices must be initialized. The device may be either a raw device or a file system device.

To illustrate the relationship of devices and databases, we created the example below. When the server was configured, we specified a device named master to contain the master database. This device was created with a size of 50 MB. The configuration process also creates two other databases on the master device, model and tempdb, each with a size of 2 MB.

The configuration process also creates a database name sybsystemprocs, and here we specified a size of 80 MB for the database, and 500 MB for a device we named device_1. We now have two devices containing databases:

- The master device is 50 MB and contains:
  - The master database: 25 MB
  - The tempdb database: 2 MB
  - The model database: 2 MB
  - Free space: 21 MB
- The device_1 device is 500 MB and now contains:
  - The sybsystemprocs database 80 MB

Now we are ready to create our first user database that we will call EMPLOYEE. We need 500 MB for this database, so we will follow these steps to accomplish this:

1. Define the device_2 device whose size is 200 MB (102400 x 2 KB)

   ```
   DISK INIT NAME = device_2, PHYSNAME = '/mnt/dbfs/employee/device_2',
   VDEVNO = 10, SIZE = 102400, DSYNC = true.
   ```

2. Create the database EMPLOYEE on the device device_1 and device_2

   ```
   CREATE DATABASE employee ON device_1=320, device2=180
   ```

You need to execute these commands in the `master` database.

We now have a database we can begin to create tables and load data. The configuration now looks like this:

- The master device of 50 MB contains:
  - The `master` database: 25 MB
  - The `tempdb` database: 2 MB
  - The `model` database: 2 MB
  - Free space: 21 MB
- The device_1 of 500 MB contains:
  - The `sybsystemprocs` database: 80 MB
  - The `employee` database: 320 MB
  - Free space: 100 MB
- The device_2 of 200 MB contains:
  - The `employee` database: 180 MB
  - Free space: 20 MB

Figure 4 illustrates this configuration.

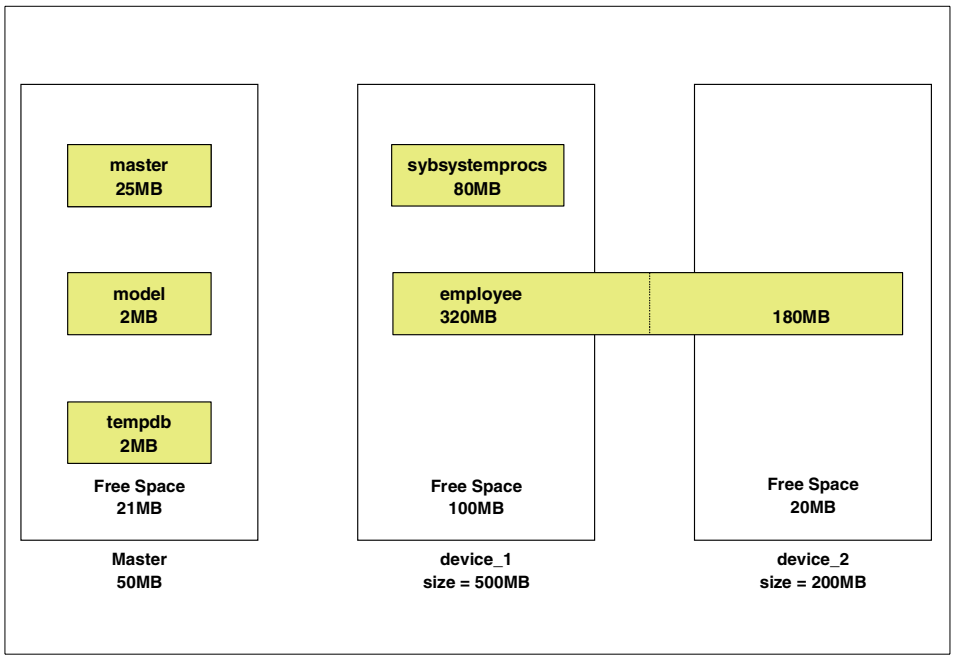

Figure 4. Sybase server with employee database

Sybase segments

The Sybase segment gives you the ability to control the placement of table data or indexes.

Databases are created using the `CREATE DATABASE` command.

Segments are assigned to devices at the database level.

When a new database is created, you will have three segments:

- logsegment
- system
- default

Figure 5 illustrates segments defined on the device `device_1` and `device_2`. As you can see, each database has three segments and a segment can span multiple devices.

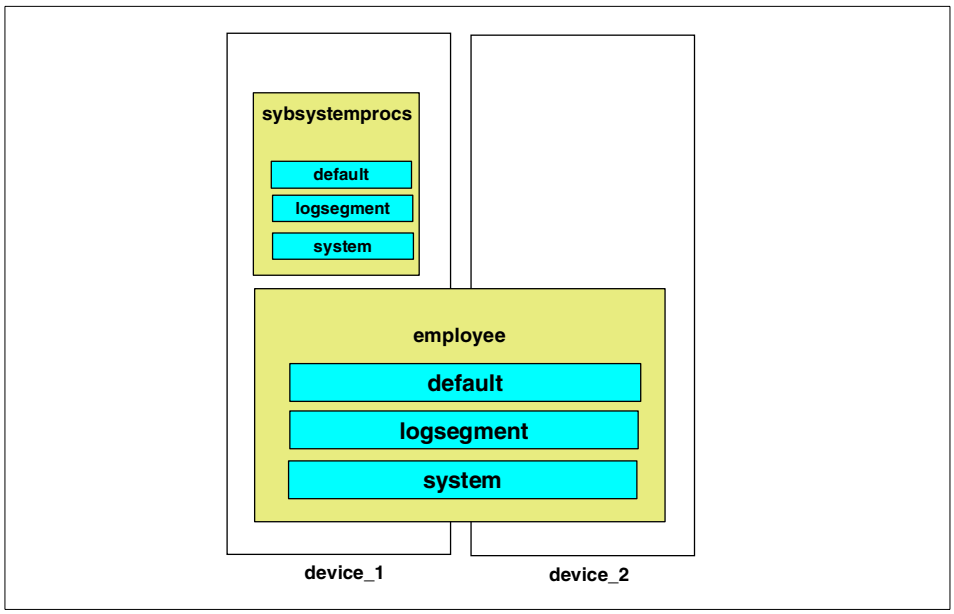

Figure 5. Segments on device_1 and device_2

Based on your requirements, you will be able to create segments which allow placing of your tables and indexes for optimum performance.

You also may have multiple segments using the same device. The configuration can be set up allow you to get the best performance. This should be planned prior to table creation.

When you have created the database, you will be able to use the stored procedure, `sp_addsegment`, to create a segment.

If you need to expand the segment for the table, you can use the `sp_extendsegment` stored procedure to include another device.

A table can be assigned to only one segment. If that segment runs out of space, there can be no further allocations for the table even though the database itself may have adequate space. You must either extend the segment to another existing device or a new device using the `sp_extendsegment`. If you want to use a new device, you need to initialize it and expand the database to the device in the following manner:

1. Execute the `DISK INIT` command to initialize a new device.
2. Execute the `ALTER DATABASE` command to alter the database on the new device
3. Execute the `sp_extendsegment` to extend the segment to the new device.

Or you can also add a new segment using the `sp_addsegment` stored procedure and direct the table to the segment using the `sp_placeobject` stored procedure. This will affect future allocations and existing data will stay as is.

This process can be accomplished without affecting user access to the database.

For purposes of illustration, we will look at our previous example. We have created our tables and started testing and found access is slower than we would like.

We can create an index for a table named `empl` on the `employee_number` column. Further, we can create this index on another device to reduce contention. The following steps should be completed:

1. Define the device `device_3` whose size is 10 MB (25600 x 2 KB).

   ```
   DISK INIT NAME = device_3, PHYSNAME = '/mnt/dbfs/employee/device_3',
   VDEVNO = 11, SIZE = 25600, DSYNC = true
   ```

2. Expand the database `employee` to the device `device_3`.

   ```
   ALTER DATABASE employee ON device_3=10
   ```

3. Remove the `default` segment from the `device_3` device so no other table data can be placed there.

   ```
   sp_dropsegment 'default', employee, device_3
   ```

4. Define a segment we will name `seg_index` on the device `device_3`.

   ```
   sp_addsegment seg_index, employee, device_3
   ```

5. Create an index `emp_index` on the `EMPL` table.

   ```
   CREATE UNIQUE NONCLUSTERED INDEX emp_index ON empl(employee_number) ON
   seg_index
   ```

Note that you must execute the `disk init` command and `ALTER DATABASE` command from the `master` database; therefore you need to execute the `use master` command before step 1.

You need to execute the `USE employee` command before step 3, because the `sp_dropsegment`, `sp_addsegment` stored procedure, and `CREATE INDEX` command should be executed for the `employee` database.

Now our configuration looks like this:

- The master device of 50 MB contains:
  - The `master` database 25 MB
  - The `tempdb` database 2 MB
  - The `model` database 2 MB
  - Free space 21 MB
- The device_1 of 500 MB contains:
  - The `sybsystemprocs` database 80 MB
  - The `employee` database 320 MB
  - Free space 100 MB
- The device_2 of 200 MB contains:
  - The `employee` database 180 MB
  - Free space 20 MB
- The device_3 of 10 MB is new and contains:
  - The `employee` database 10 MB (for index)
  - No free space

Figure 6 shows this configuration.

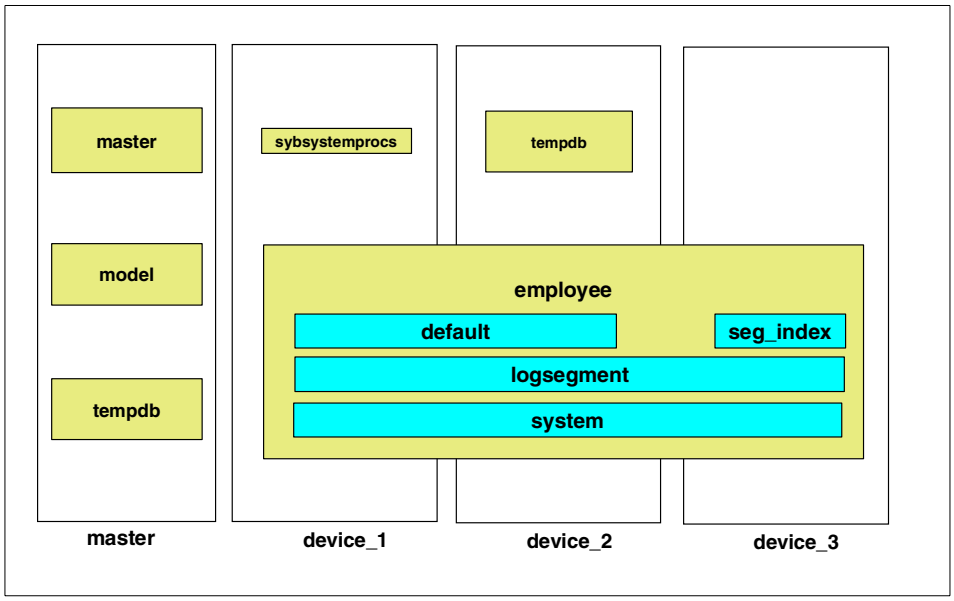

Figure 6. Sybase Server with employee database segments

Sybase tables

When a table is created using the `CREATE TABLE` command, the table may be mapped to a specific segment, or if no segment is specified, the table is placed on the `default` segment.

If the table already exists and is using only the `default` segment, the `sp_placeobject` stored procedure may be used to direct future allocations for the table to a new segment.

Indexes may also be directed to specific segments. One of the more practical uses of segments is to allow separation of data and nonclustered indexes.

4.1.1.4 Transaction logs

Sybase transaction logging is accomplished at the database level. When you create a database, there will be a table `syslogs` which is used for transaction logging and recovery for the database.

The `syslogs` table will be mapped to the `logsegment` segment mentioned above.

When you create a new database, you have an option to place the `syslogs` table and the `logsegment` segment on a separate device. The table `syslogs` is always created on an internal segment, `logsegment`. When you separate data and logs, you have a greater chance of recovery if a device goes down. You accomplish this by specifying the `log on` option of the `CREATE DATABASE` command. Having a separate log device also allows you to more easily mirror the log device to provide even better recovery.

If the `logsegment` fills, processing will be suspended until the situation is cleared.

You can dump the transaction log to disk or tape devices using the Sybase Backup server.

The transaction log can also be truncated using the with `TRUNCATE_ONLY` option of the `DUMP TRAN` command. This option, however, will leave you in a position of losing data if your server should go down and you have to restore from a backup. There is some protection for this since it will not allow another transaction dump to process until a complete database dump has successfully completed.

4.1.2 DB2 database structure

The instance is the logical database server in the DB2 UDB environment. A DB2 instance can manage multiple databases.

A DB2 database consists of table spaces to which tables are assigned. Table spaces are created on containers. Figure 7 shows this relationship.

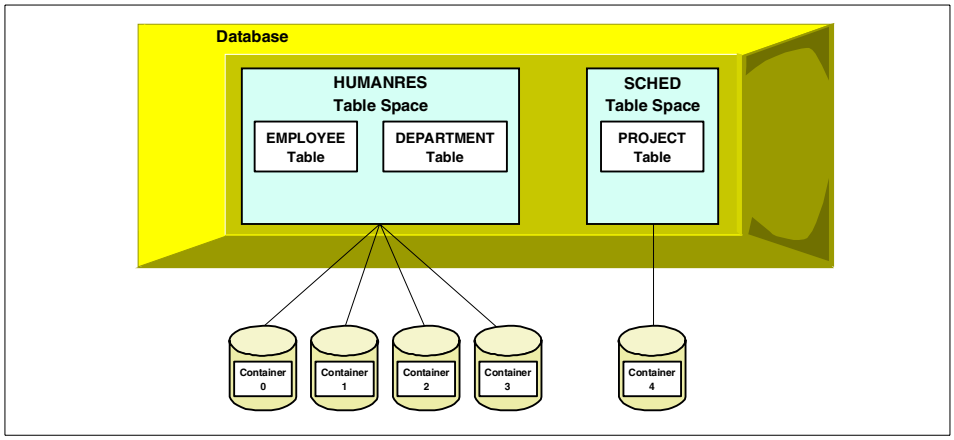

Figure 7. Table spaces and containers

4.1.2.1 DB2 table space

A table space can be described as the layer between the physical device, the container, and the table which contains the data. Careful planning in the process to create table spaces can have positive impact on the database. When you create a DB2 UDB database, the following three table spaces are created:

- SYSCATSPACE
- TEMPSPACE1
- USERSPACE1

The SYSCATSPACE table space contains the system catalogs.

The TEMPSPACE1 table space is used by the database manager for SQL operations, holding transient data like intermediate tables during sorts, reorganizing tables, creating indexes and joining tables.

The USERSPACE1 table space is used to store user tables. You may create your own tables in the USERSPACE1 table space using your own names.

Once you have created a database, you may create your own table spaces to store user tables. The USERSPACE1 table space may be dropped if you have created your own table spaces.

If you want to use declared global temporary tables, you have to create a user temporary table space after the database is created. Declared global temporary tables will be discussed in “Declared temporary tables” on page 142.

You can perform the DB2 UDB backup and recovery at the table space level, therefore if we plan properly, we will place tables with similar backup and recovery requirements on like table spaces, or place larger tables on a single table space to allow recovery at the table level.

When planning your table spaces, there are two types of table spaces that can be created; System Managed Space (SMS) and Database Managed Space (DMS).

4.1.2.2 DB2 container

A container is a physical storage device, identified by a directory name, a device name, or a file name. Containers are normally set up by your AIX or UNIX system administrator and are raw devices or file systems. Once set up, the CREATE TABLESPACE or ALTER TABLESPACE commands create or make them usable to DB2 UDB.

It is important to note that DB2 table space containers are written to in a round-robin manner. The database manager writes an extent, which is a contiguous allocation of space within a table space, on the first container in the table space, then the second, third, and continues until it wraps back around to the first.

4.1.2.3 SMS table spaces

In an SMS table space, the operating system's file system manager allocates and manages the space where your table is to be stored. The user decides where the files are located while DB2 UDB controls the file names and the file system manages and allocated space for them. For SMS table spaces, each container would be a directory in the Operating System file system.

4.1.2.4 DMS table spaces

If you use DMS table spaces, the database manager will control the storage space. The database administrator will decide which devices to use and DB2 UDB manages the space on those devices. The table space definition will include a list of devices or files assigned to the table space. Each container will be either a fixed size Operating System file or a raw device.

4.1.2.5 DB2 table space types

There are four types of table spaces:

- Regular table spaces

  Tables containing user data are created on `regular` table spaces. When we execute the `CREATE DATABASE` command, a table space named `USERSPACE1` is created. You may optionally create other table spaces using your own names.

- Long table spaces

  When a table has long field columns (`LONG VARCHAR or LONG VARGRAPHIC`) or large object columns (`CLOB, BLOB` or `DBCLOB`), those columns data may be placed on a long table space separately. Long table spaces must be DMS table spaces.

- System temporary table spaces

  The database manager must have at least one temporary table space defined. The `CREATE DATABASE` command creates a temporary table space named `TEMPSPACE1`. You can create your own temporary table space using any valid name and then drop this table space if you like.

- User temporary table spaces

  You must create user temporary table spaces to use declared global temporary tables. See 7.4, “Declared temporary tables” on page 142 for more details.

  Temporary table spaces can be either SMS or DMS.

4.1.2.6 DB2 data files

The DB2 physical structure of the data files is dependent on the type of table space you define. As mentioned above, there are two types of table spaces and both types can be utilized within a single database. They are:

- System managed space (SMS) table space

  The operating system’s file manager allocates and controls the storage space within the file system.

- Database managed space (DMS) table space

  The database manager controls the storage space within a logical volume or operating system file.

When you create an SMS table space, all data files are located under the database subdirectories, or you may specify the directory path(s) where you would like to store the data. The file names are selected by DB2 automatically when created. The number of subdirectories to be created are specified at table space creation time and cannot be changed later.

SMS table space

The database administrator can specify any operating system directory accessible by the system for SMS containers. Each directory will be a container and one table space can have multiple containers. With this in mind the maximum size of a table in an SMS table space will be determined by the number of containers specified, multiplied by the maximum file size supported by the operating system.

Note

There are architectural limits of the maximum table size in a table space depending on the data page size you specify when creating the table space. See Appendix A, “SQL Limits” in the *SQL Reference*, SC09-2975, for more information.

Table 3 lists the files that can be found in the SMS table space directory.

Table 3. Files in an SMS table space container

File name	Purpose
SQLxxxxx.DAT	Table file where all rows of a table are stored except LONG VARCHAR, LONG VARGRAPHIC, CLOB, BLOB and DBLOB data.
SQLxxxxx.LF	These files contain the data types LONG VARCHAR or LONG VARGRAPHIC data. These tables are only created if LONGVARCHAR or LONG VARGRAPHIC columns exist in the table.
SQLxxxxx.LB	These files are created only if CLOB, BLOB or DBLOB columns appear in the table.
SQLxxxxx.LBA	Files containing allocation and free space information about the SQLxxxxx.LB files.
SQLxxxxx.INX	These files contain index files for the tables. All indexes for a corresponding table are stored in this file. It is only created if indexes are defined. When indexes are dropped, the space is not freed until all indexes for the table have been dropped.
SQLxxxxx.EIX	Damaged SQLxxxxx.INX files would be stored here.
SQLxxxxx.DTR	Temporary files for a REORG of an SQLxxxxx.DAT file.
SQLxxxxx.LFR	Temporary file for a REORG of an SQLxxxxx.LF file.
SQLxxxxx.RLB	Temporary file for a REORG of an SQLxxxxx.RL file.
SQLxxxxx.RBA	Temporary file for a REORG of an SQLxxxxx.RB file.

DMS table space

For DMS table spaces, the database management system becomes responsible for managing the space. DMS table spaces are built on pre-allocated raw devices or files. When you create a DMS table space, you need to specify its containers definition including the container type (device or file), which devices or files to use, and the size of the each container.

The directories on which you intend to create DMS container files, or raw devices need to have the owner and group set to match the DB2 instance owner and group.

Table objects for the data, indexes, and LONG data columns can all be stored in the same table space or in different table spaces. The size of the DMS table spaces can be increased by adding more containers. For DMS table spaces, a container can be either a file system file or a raw device.

The DMS table space is implemented in a similar fashion to the Sybase device. Table 4 compares the SMS, DMS and Sybase Device concept.

Table 4. Comparing SMS, DMS, and Sybase disks

	SMS table space	DMS table space	Sybase device
Tablespaces can share containers (or devices)	NO	NO	YES
Allocate space as needed	YES	NO	NO
Increase number of containers in table space	NO	YES	YES
Store index data in separate space	NO	YES	YES
Store long data in separate space	NO	YES	NO
One table (index, data, LOB) can span several table spaces	NO	YES	YES (Index and data can be stored separately)
Container may be a raw device	NO	YES	YES

4.1.2.7 DB2 tables

As with Sybase, DB2 UDB tables are created using the `CREATE TABLE` command. Tables are created on table spaces and can use the `USERSPACE1` table space, or another table space you have created. The regular table must be created in a regular table space. Declared temporary tables must be created in a user temporary table space.

For DMS table spaces, DB2 UDB allows you to create your table and indexes on different table spaces. You may also direct long field data or large object data to a long table space.

4.1.2.8 DB2 logging

DB2 UDB log files are created for each database. As transactions are processed, they are written first to the log file and the database is updated at a later time. DB2 UDB offers two types of logging, circular and archival. There are also two types of log files, primary and secondary.

The type of logging you use will be determined by the type of recovery that is best suited for your environment.

Circular logging is the default type of logging and supports non-recoverable databases. Crash recovery and version recovery are available, but roll-forward recovery is not available using this method.

Archival logging supports crash recovery, version recovery, and roll-forward recovery. For archival logging, log files are archived when they become inactive. You must specify `LOGRETAIN ON` to configure a database to perform archival logging. Specifying `USEREXIT ON` will allow a user exit program to move log files to archival directories, devices or Tivoli Storage Manager (formerly ADSM) for retention.

Primary log files are pre allocated during the first connection to the database. They establish a fixed amount of storage allocated to the recovery log files. The database configuration parameter `LOGPRIMARY` determines the number of primary files created, and `LOGFILSIZ` determines the size of each file.

Secondary log files are allocated one at a time, as needed, when the primary log file becomes full. The number of secondary log files will depend on the value specified in the configuration parameter, `LOGSECOND`, and the size of each secondary log file is specified based on the `LOGFILSIZ` parameter.

4.1.2.9 DB2 UDB database directories

When a database is created, DB2 creates a separate subdirectory to store control files (such as log header files) and to allocate containers to default table spaces. Objects associated with the database are not always stored in the database directory but can be stored in various locations, even devices.

The database is created in the file system path specified using the `ON` parameter of the `CREATE DATABASE` command. The naming scheme used on UNIX based systems is:

```
path/$DB2INSTANCE/NODEnnnn/SQL00001
```

Where:

- `path` is the user specified location to crate the database
- `$DB2INSTANCE` is the name of the instance you attached to. If you have not explicitly attached to any instances using the `ATTACH` command, the value specified in the `DB2INSTANCE` environment variable will be used.
- `NODEnnnn` is the node identifier in a partitioned database environment.The first node will be `NODE0000`. Unless you are using DB2 UDB Enterprise-Extended Edition, this is always `NODE0000`.

SQL00001 contains the objects associated with the first created, and subsequent databases created will be SQL00002, 3, through n.

4.1.2.10 Database files

The files shown in Table 5 will be associated with the creation of each database.

Table 5. DB2 files

FILE	Description
SQLDBCON	This file stores the tuning parameters and flags for the database.
SQLOGCTL.LFH	This file is use to help track and control all of the database log files.
Syyyyyy.LOG	These are database log files, beginning with 000000 and going through 99999. The number of these files is determined by the `LOGPRIMARY` and `LOGSECOND` database configuration parameters. The size of the files is determined by the `LOGFILSIZ` parameter. With circular logging, the files are reused with the same numbers. For archive logging, the file number will increment as logs are archived and new logs created. When the number reaches 999999, it will wrap around to 000000.
SQLINSLK	This file helps insure that the database is used by only one instance of the database manager.
SQLTMPLK	This file also helps to insure that the database is used by only one instance of the database manager.
SQLSPCS.1	This file contains the definition and current state of all table spaces in the database.
SQLSPCS.2	This is a backup copy of sqlspcs.1 either this file or sqlpscs.1 must be available to access this database.
SQLBP.1	This file contains the definition of all buffer pools in the database.
SQLBP.2	This is a copy of SQLBP.1. This file or SQLBP.2 must be available to access this database.
DB2HIST.ASC	This is the database history file. It maintains a history of administrative operations to the database, such as backups and restores.
DB2HIST.BAK	This is a back up copy of `DB2HIST.ASC`.

These files are critical to the operation to the database management system and should not be changed or deleted.

4.1.3 DB2 logical storage

Although we discussed table spaces previously, it was directed more toward physical use. Here, we will look at table spaces as the logical use for containers.

Within Sybase Adaptive server, we have devices that are in many ways similar to the DB2 table space. Once a device is set up in Sybase, using the `DISK INIT` command, you must then associate the device with a database using either the `CREATE DATABASE` command or `ALTER DATABASE` command. Once these steps are accomplished, the device is ready to use as the `DEFAULT` segments for that database. The Sybase server will now manage the space for that device. In this usage, the Sybase device performs the functions of both the container and the table space.

As we discussed earlier in the physical design sections, Sybase has another mechanism to manage storage space, and that is the segment. A segment is really a mechanism to direct tables or indexes to a specific device for performance considerations. A segment can be assigned to only one device when created, but can be extended to other devices using the `sp_extendsegment` stored procedure. In this scenario, the Sybase segment will be the same as the DB2 table space defined to a specific container. The segment also applies to the entire device, not a part of a device in this scenario.

Now the similarities end, since multiple segments can point to the same device. Therefore, if we look at each in RDBMS terminology, the Sybase device and the segment have a many-to-many relationship, where the DB2 container and table space have a many-to-one relationship.

Proper use of segment in the Sybase server can be very beneficial both in the space utilization for devices as well as performance. A table and its index may reside on different segments, and if a table has multiple indexes, the table and each index can reside on a different segment.

In Sybase:

- Devices contain segments.
- Devices may contain more than one segment.
- Segments can span multiple devices.
- Segments must reside on at least one device.
- Tables are created on segments.
- Indexes can be on different segments.
- Each index can be on a different segment.

The DB2 UDB table space is the logical relationship between a container and the data. Within DB2, there is a rule of hierarchy, but not in Sybase. Table spaces are created in databases, and a table space uses containers. Tables and indexes are created in table spaces.

In DB2 UDB:

- Containers can belong to only one table space.
- Table spaces are created on containers.
- Table spaces can span multiple containers.
- Table spaces must have at least one container.
- Tables are created in table spaces.
- For DMS table spaces, a table, its indexes, and its LONG data can be assigned different table spaces.

For DMS table spaces, additional containers can be added as needed. SMS has a fixed number of containers at creation time and the number cannot be increased.

4.1.4 Tables

In Sybase, when you create a table, you can specify that the table be created on a specific segment. The following statement will create a table named `empl` on the segment `segment_1`:

```
CREATE TABLE empl (employee_number CHAR(6),
                   employee_name VARCHAR(30))
        ON segment_1
```

You can create indexes separately using the `CREATE INDEX` command and it also has the option to create the index on a specific segment. If no segment name is provided in either the `CREATE TABLE` or `CREATE INDEX` commands, they will be created on the `default` segment.

For DB2 UDB the `CREATE TABLE` command is used to create the table:

```
CREATE TABLE empl (employee_number CHAR(6),
                   employee_name VARCHAR(30))
        IN TABLESPACE tablespacea,
         INDEX IN tablespaceb
```

This example will create the DB2 table `empl` in the table space `tablespacea`. It will place the indexes on the table `EMPL` in the tablespace `tablespaceb`.

4.2 Data type comparisons

The purpose of this section is to describe the data types supported by Sybase and DB2 UDB. We list the data types by database management system, and then identify the differences as well as the modifications necessary to port from Sybase to DB2 UDB. Data types supported by DB2 only are not listed.

Table 6 lists the data types supported by the Sybase database management system. Since our focus is on conversion, we will list and discuss only those data types supported by Sybase.

Table 6. Sybase data types supported

Data Type	Description	Range	Bytes of storage
tinyint	whole numbers	0-255 (no negative allowed)	1
smallint	whole numbers	2^{15} -1 (32767) to -2^{15} (-32768)	2
int	whole numbers	2^{31} (2,147483647) to -2^{31} (2,147483648)	4
numeric(*p, s)*	numeric with decimal numeric with scale of 0 displayed without decimal point	10^{38} -1 to -10^{38}	2 to 17
decimal(*p, s*)	numeric with decimal	10^{38} -1 to -10^{38}	2 to 17
float(precision)		machine dependent	4 or 8 [1]
double precision		machine dependent	8
real		machine dependent	4
smallmoney	monetary values	214,748.3647 to -214,748.3548	4
money	monetary values	922,337,203,685,477.5807 to -922,337,303,685,477.5808	8
smalldatetime	date	January 1, 1900 to June 6, 2079 [2]	4
datetime	date	January 1, 1753 to December 31, 9999 [3]	8

Data Type	Description	Range	Bytes of storage
char(n)	fixed length data in single byte character sets	1 to 255	n
varchar(n)	variable character	1 to 255	actual entry length
nchar(n)	national character	1 to 255	actual entry length
nvarchar(n)	national variable character	1 to 255	actual character length
text(n)	varaible length printable character data	2^{31} -1 (2,147,483,647)	0 or multiples of 2K
binary(n)	variable length binary data	255 bytes or less	n
varbinary(n)		255 bytes or less	actual entry length
image		2^{31} -1 (2,147,483,647)	0 or multiple of 2K
bit	used for true/false conditions	0 or 1	1

[1]Float storage is 4 bytes if precision <16 or 8 bytes if precision is >=16.

[2]Smalldatetime values are accurate to the minute. Storage size is 4 bytes: 2 bytes for the number of days since January 1, 1900, and 2 bytes for the number of minutes since midnight.

4.2.1 Character data types

This section discusses the character data types supported by Sybase and DB2 and a comparison of functionality. First we show those data types supported by Sybase with a corresponding DB2 data type. Table 7 lists the corresponding data types and shows differences in content or functionality.

If a data type is not listed for DB2, we then list a suggested data type to use. Some of these differences are minor, and the data type to use will depend on your data content.

Table 7. Character data types

Sybase Data Type	Bytes of storage	Comments	USE DB2 Data Type	Bytes of storage	Comments
char(n)	n	1 to 255 characters	char(n)	b	1 to 254 characters
varchar(n)	entry length	1 to 255 characters	varchar(n) [3]	entry length	1 to 32,762 characters
nchar(n) [1]	n * char size	1 to 255 characters	graphic(n) [3]	2 * n	1 to 127 characters
nvarchar(n)[1]	entry length " char size	1 to 255 characters	vargraphic(n) [3]	2 * entry length	1 to 2,000 characters
text [2]	0 or multiple of 2K	1 to 2,147,483,647	CLOB(n)		1 to 2,147,483,647

[1]This data type allows national character set specifications, where 1 character uses > 1 byte.

[2] For the text data type for Sybase, the data is stored in increments of a data page (2K bytes), whereas CLOB stores up to the maximum number of characters specified by (n). Sybase also allows you to search for data in text columns using the `LIKE '%DATA'` option in the where clause. You will need to check your applications for this.

[3] Special restrictions apply to an expression resulting in a varying-length string data type whose maximum length is greater than 254 bytes. Such expressions are not permitted in a `SELECT DISTINCT` statement's `SELECT` list, a `GROUP BY` clause, an `ORDER BY` clause, a column function with `DISTINCT`, and a sub-select of a set operator other than `UNION ALL`.

These are the data types that fit best; however, the contents of your data may require a different type substitution.

4.2.2 Numeric data type

Table 8 compares the Sybase numeric data types and the DB2 data types. The primary differences here are that the `tinyint` data type is not supported by DB2, so you will use the `smallint` data type. The `tinyint` data type does not allow negative values, so this will be data dependent. Also, there are differences in the `float` and `real` data types.

Table 8. Numeric data types

Sybase data type	Bytes of storage	Comments	Use DB2 data type	Bytes of storage	Comments
tinyint	1	value of 0 to 255 no negatives	smallint	2	-32768 to 32767
smallint	2	-32768 to 32767	smallint	2	-32768 to 32767
int	4	-2^{31} to 2^{31} -1	int	4	-2^{31} to 2^{31} -1
			bigint	8	-9223372036854775808 to +9223372036854775807
numeric(p, s)	2 to 17	-10^{38} to 10^{38} -1	numeric(p, s)	(p/2)-1	-10^{38} to 10^{38} -1
decimal(p, s)	2 to 17	-10^{38} to 10^{38} -1	decimal(p, s)	(p/2)-1	-10^{38} to 10^{38} -1
real	4	machine dependent	real	4	zero or can range from -3.402E+38 to -1.175E-37, or from 1.175E-37 to 3.402E+38.
float(p)	4 or 8[1]	machine dependent	float or double	8	zero or can range from -1.79769E+308 to -2.225E-307, or from 2.225E-307 to 1.79769E+308.

Sybase data type	Bytes of storage	Comments	Use DB2 data type	Bytes of storage	Comments
double precision	8	machine dependent	float or double	8	zero or can range from -1.79769E+308 to -2.225E-307, or from 2.225E-307 to 1.79769E+308.

[1]In Sybase, float storage is 4 bytes if precision <16 or 8 bytes if precision is >=16

4.2.3 Datetime data type

There are some differences between Sybase and DB2 in the date and time data types. Table 9 shows the differences. The primary differences are that DB2 does not support the `smalldatetime` data type, so you need to use the `timestamp` type. The differences in the `datetime` and `timestamp` are the format and the range of dates.

Table 9. Datetime data types

Sybase data type	Bytes	Comments	Use DB2 data type	Bytes	Comments
smalldatetime	8	January 1, 1900 to June 6, 2079	timestamp	10	January 1, 0000 to December 31, 9999
datetime	8	January 1, 1753 to December 31, 9999	timestamp	10	January 1, 0000 to December 31, 9999
			date	4	January 1, 0000 to December 31, 9999
			time	3	00:00:00 to 24:59:59

In Sybase, smalldatetime values are accurate to the minute. Storage size is 4 bytes: 2 bytes for the number of days since January 1, 1900; and 2 bytes for the number of minutes since midnight.

In Sybase, datetime values are accurate to 1/300 of a second on platforms that support this level of granularity. Storage size is 8 bytes: 4 bytes for the number of days since the base date of January 1, 1900, and 4 bytes for the time of day.

In DB2 UDB, timestamp values are accurate to the microsecond. Storage size is 10 bytes.

There are also a number of date routines provided with Sybase, as well as DB2, and they will be discussed in detail in Chapter 7, “Application conversion” on page 115. DB2 provides the functionality for most of these routines but be careful of format differences.

Sybase has multiple formats by which the date can be returned in a select statement, using the `CONVERT` function, or `SELECT GETDATE(N)` to use the date functions.

The default format for Sybase datetime and smalldatetime is as following:

- Jan 1 2000 12:00AM

This is midnight, January 1, 2000.

The default format for DB2 are:

- TIMESTAMP is YYYY-MM-DD-HH-MM-SS-NNNNNN.
- DATE is MM-DD-YYYY
- TIME is HH-MM-SS

You can change the display format using the `CHAR` function. See 7.3.1.2, “Functions that have different names” on page 129.

4.2.4 Binary data type

Table 10 lists the binary data types supported by Sybase and the possible DB2 data types. Since bit data type values are 0 or 1, use the `CHAR(1) FOR BIT DATA`. Here, `FOR BIT DATA` specifies that the contents of the column are to be treated as bit (binary) data.

Table 10. Binary data types

Sybase data type	Bytes of storage	Comments	Use DB2 data type	Bytes of storage	Comments
binary	n	0 to 255 bytes	character(n) for bit data	n	1 to 254 characters
varbinary	entry length	0 to 255 bytes	varchar(n) for bit data	entry length	1 to 254 characters
image		2^{31} bytes or less stored in multiples of a page	BLOB(n)	entry length	1 to 2^{31} bytes. Actual data length will be stored
bit	1	0 or 1	char(1) for bit data	1	1 byte

There are some differences in the way the binary(n) data type and the char for bit data types are handled. Sybase will accept either character data or numeric data for the binary(n) data type. See the following examples:

```
1> create table test05 (col1 binary(10))
2> go
(1 row affected)
1> insert test05 values(1024)
2> go
(1 row affected)
1> insert test05 values('1024')
2> go
(1 row affected)
1> insert test05 values(0x1024)
2> go
(1 row affected)
1> select * from test05
2> go
 col1
 ----------------------
 0x00000400000000000000
 0x31303234000000000000
 0x10240000000000000000

(3 rows affected)
```

Here we created a table with a binary data type with a length of 10, inserted a numeric value of 1024 into the table, inserted a character value of '1024', and then inserted a binary value of 0x1024. Notice that the numeric value is converted and the character is not. Also, the padding for both types is binary zero(0), and the select presents the data as 0x, then the data values.

In DB2, you cannot insert numeric values into a char for bit data type column. See the following examples:

```
$ db2 "create table table05 (col1 char(10) for bit data)"
DB20000I The SQL command completed successfully.
$ db2 "insert into table05 values ('1024')"
DB20000I The SQL command completed successfully.
$ db2 "insert into table05 values (x'1024')"
DB20000I The SQL command completed successfully.
$ db2 "select * from table05"

COL1
----------------------
x'31303234202020202020'
x'10242020202020202020'

2 record(s) selected.
```

DB2 treats both inserts in the same manner as Sybase, but instead of padding with zero, it pads with spaces.

Notice also that the output of the select statement is prefixed with an x and enclosed in single quotes. When the inserted values contain the full length of 10 characters, then the data selected is the same except for the 0x for Sybase and x' for DB2 UDB.

4.2.5 Other data types

Table 11 lists other data types supported by Sybase. The `MONEY` and `SMALLMONEY` data types are not supported by DB2, so we will use NUMERIC(p, s). These can specify 2 or 4 decimal positions, depending on your needs.

Table 11. Other data types

Sybase data type	Bytes of storage	Comments	Use DB2 data type	Bytes of storage	Comments
smallmoney	4	-214,748.3548 to 214,748.3647	numeric(10, 4)	6	-999,999.999 to 999,999.9999
money	8	-922,337,203,685, 477.5808 to 922,337,203,685, 477.5808	numeric(19, 4)	10	-999,999,999,999, 999.9999 to 999,999,999,999, 999.9999
identity		see paragraph below	identity		
user data types		see paragraph below	user defined types		

4.2.5.1 Identity columns

In this section we will discuss the Sybase identity and the DB2 identity data types and will explain their differences.

Sybase identity columns

Sybase allows a user to specify a data type of `IDENTITY` for a column which will generate a unique sequential number for each row in the table. `IDENTITY` columns have a data type of `NUMERIC` with scale of 0. The format for creating an `IDENTITY` column is:

```
CREATE TABLE sample_table (id_col NUMERIC(6, 0) IDENTITY)
```

The system will generate a sequential number for each insert into the table. The `IDENTITY` cannot be updated by the user application and cannot contain nulls. It may, however be referenced by using the `@@SYB_IDENTITY` global variable.

You can get the information about `IDENTITY` columns by executing the following command:

```
SELECT @@IDENTITY
```

DB2 identity columns

DB2 supports the `IDENTITY` data type in a very similar manner. DB2 will generate an `IDENTITY` column which will be incremental for each insert and will not allow nulls. There are some additional options which allow a starting number at creation time, as well as an option to determine the value to increment.

For example, INCREMENT BY 2 would cause this column to be incremented by two for each row inserted. The INCREMENT BY value may be negative, to cause the value to decremented if you wish to start with a high number and decrement for each insert. Formats for the CREATE TABLE commands are very similar:

```
CREATE TABLE sample_table (id_col NUMERIC(6, 0) GENERATED ALWAYS AS
IDENTITY)
```

Or:

```
CREATE TABLE sample_table (id_col NUMERIC(6, 0) GENERATED ALWAYS AS
IDENTITY (START WITH 900000,INCREMENT BY -1))
```

The GENERATED ALWAYS option indicates that DB2 will always generate a value for the IDENTITY column when a row is inserted into the table, and this is the same behavior as Sybase's IDENTITY column.

To obtain the last IDENTITY column value, DB2 has the IDENTITY_VAL_LOCAL() function. We will discuss the IDENTITY_VAL_LOCAL() function in "The @@identity global variable" on page 151.

When you are converting a Sybase table with an IDENTITY column to DB2 table, and you wish to retain the IDENTITY column value from the source Sybase table, specify the last identity value plus 1 with the START WITH option of the CREATE TABLE statement. Otherwise, after loading the data into the target DB2 table, a future insert may generate a duplicate identity value. This is because the MODIFIED BY IDENTITYOVERRIDE option of the DB2 LOAD utility, which you need to specify to make the DB2 LOAD utility get the identity values from the input file instead of generating them, does not keep track of the values. Therefore, the first insert after the loading will generate the identity value which is specified with the START WITH option of the CREATE TABLE statement.

4.2.5.2 User defined data types

Both Sybase and DB2 allow user defined data types based on existing data types.

Sybase user defined data types

For Sybase, user defined data types are added to the database by the stored procedure sp_addtype as in the following example:

```
sp_addtype mytype, DECIMAL(8,2)
```

You may use the stored procedure sp_help to get information about an existing user defined data type.

When adding a user defined data type, you must be in the database where you want the new user defined data type to exist. You can, however add user defined data types to the `MODEL` database and any new databases created will contain your new user defined data types.

You can bind RULES with the user defined data type definitions using the `sp_bindrule` stored procedure. When you use the user defined data type in table column definitions, these columns inherit the rules, properties and defaults associated with that user defined type.

DB2 user defined data types

DB2 also supports user defined data types. Creation of user defined distinct type with DB2 is accomplished using the following `CREATE` statement:

```
CREATE DISTINCT TYPE mytype AS DECIMAL(8,2) WITH COMPARISONS
```

Note that you *cannot* compare a user defined data type data with its base data type data directly. This concept is known as strong data typing. DB2 provides strong data typing to avoid end-user mistakes during the assignment or comparison of different types of real-world data. For example, the following select statement will give an error condition because the column `salary` is defined as data type `mytype`.:

```
CREATE table mytable (id INT, salary MYTYPE)
SELECT * FROM mytable WHERE salary > 35000
```

To compare the salary column with the constant value 3500, you should modify the select statement as following:

```
SELECT * FROM mytable WHERE salary > CAST (35000 AS mytype)
```

Or:

```
SELECT * FROM mytable WHERE salary > mytype(35000)
```

Or:

```
SELECT * FROM mytable WHERE DECIMAL(salary) > 35000
```

In Sybase, user defined data types do not support strong data typing, and you can compare a user defined data type data with its base data type data directly. Therefore, when you convert your applications, look at SQL statements carefully and use casting functions to handle user defined data types if necessary.

4.2.5.3 Host variable declarations

When dealing with data types, we not only have to look at the SQL data types for Sybase and DB2, but also with the programming language data types. When converting we need to insure that the new data types are compatible with the DB2 precompiler. Table 12 shows a comparison between Sybase, DB2 UDB, and the C or C++ languages.

Table 12. SQL and C/C++ data type comparison

Sybase data type	Db2 data type	C data type declaration
char(n)	CHAR(n)	char char_var[n+1];
varchar(n)	VARCHAR(n)	struct{short len; char data[n]} varchar_var;
text	CLOB(n)	SQL TYPE IS CLOB(n) v-name;
numeric(p, s)	NUMERIC(p, s)	Double num_var;
decimal	DECIMAL	Double dec_var;
money, smallmoney	DECIMAL(p, s)	Double dec_car;
integer	INTEGER	Long int int_var;
smallint	SMALLINT	Short int_var;
bit	CHAR(n) for bit data type	Char;
image	BLOB(n)	SQL TYPE is BLOB(n) v_name;
datetime	TIMESTAMP	Char tms_var[27];
datetime(date only)	DATE(YYYY-MM-DD)	Char dt_var[11];
datetime(time only)	TIME (HH:MM:SS)	Char tm_var[11];

Chapter 5. Database conversion

In this chapter we cover the creation of databases on the DB2 UDB system, including conversion methods, creation of DB2 UDB instances, databases, table spaces, tables, data types, tables, views, and indexes. We also look at security in the Sybase database and corresponding DB2 UDB security implementation.

Actual data conversion and population of the tables will be covered in Chapter 6, “Data conversion” on page 91.

5.1 Conversion method

Our first task in the conversion process is to decide how the conversion process should be accomplished. We will look at manual conversion with no tools as well as conversion using conversion tools.

We must first plan for the conversion by taking inventory of the existing database on the Sybase server. Here are some of the things we need to consider:

- Size of the database
- Number of tables
- How our data is accessed
- If the current database uses segments
- Indexes
- Data types, with special attention to user defined data types
- Security on tables and other objects

To get started, we will use the Sybase `sp_helpdb` stored procedure to show us information about the database to be converted. The `sp_helpdb` stored procedure with no parameters shows information about all databases. If you provide a parameter with the database name, you will see information only about the database we are going to convert, which is named `test1`. Several command and output examples are shown in the following sections.

As shown in Figure 8, the stored procedure `sp_helpdb` with no parameters will display information about all databases in the system.

```
1> sp_helpdb
2> go
 name                    db_size       owner                   dbid
        created
        status

 ----------------------- ------------- ----------------------- ------
       --------------
       ------------------------------------------------------------------------
-----------------------------
master                          6.0 MB sa                            1
        Jan 01, 1900
        no options set

 model                           2.0 MB sa                            3
        Jan 01, 1900
        no options set

 pubs2                           3.0 MB sa                            4
        Aug 02, 2000
        trunc log on chkpt

 sybsystemdb                     2.0 MB sa                        31513
        Nov 03, 1999
        no options set

 sybsystemprocs                 80.0 MB sa                        31514
        Aug 02, 2000
        trunc log on chkpt

 tempdb                          2.0 MB sa                            2
        Aug 03, 2000
        select into/bulkcopy/pllsort
 test1                         15.0 MB sa                             5
        Aug 03, 2000
        trunc log on chkpt

1 row affected)
(return status = 0)
1>
```

Figure 8. sp_helpdb output

The next screen, Figure 9, shows information on the database we want to convert, `test1`.

```
1> sp_helpdb test1
2> go
 name                 db_size       owner                  dbid
        created
        status

 -------------------- ------------- ---------------------- ------
       --------------
       ------------------------------------------------------------------------
------------------------------
 test1                          15.0 MB sa                          5
        Aug 03, 2000
        trunc log on chkpt


(1 row affected)
 device_fragments              size          usage
        free kbytes
 ----------------------------- ------------- --------------------
       ----------------
 device_1                      15.0 MB       data and log
                    7614
 device
        segment

 ------------------------------
       ------------------------------------------------------------------------
------------------
 device_1
        default

 device_1
        logsegment

 device_1
        system

(return status = 0)
1>
```

Figure 9. sp_helpdb output using database name option

Here we see that our database name is `test1`. It is 15 MB in size, it is all on one device named `device_1`, and the log segment and data share the same device. We will use this information when planning to create our DB2 table spaces.

Next we will look at the segments defined within database `test1` using the stored procedure `sp_helpsegment`. See the output example shown in Figure 10.

```
1> use test1
2> go
1> sp_helpsegment
2> go
 segment name                          status
------- ------------------------------ ------
       0 system                              0
       1 default                             1
       2 logsegment                          0
(return status = 0)
```

Figure 10. sp_helpsegment output

The output from the `sp_helpsegment` stored procedure tells us there are no user segments defined for this database.

To look at the tables, stored procedures and other objects we will use the `sp_help` stored procedure.

To view table information, use stored procedure `sp_help table_name`. See the output example shown in Figure 11.

```
1> sp_help table01
2> go
 Name                           Owner
        Type
 ------------------------------ ------------------------------
       ----------------------
 table01                        dbo
        user table

(1 row affected)
 Data_located_on_segment        When_created
 ------------------------------ --------------------------
 default                               Sep 19 2000  1:07PM
 Column_name     Type            Length Prec Scale Nulls Default_name
        Rule_name       Identity
 --------------- --------------- ------ ---- ----- ----- ---------------
       --------------- --------
 col1            varchar             40 NULL  NULL     0 NULL
        NULL                    0
 col2            varchar             10 NULL  NULL     0 NULL
        NULL                    0
 col3            float                4 NULL  NULL     0 NULL
        NULL                    0
index_name           index_description
        index_keys
index_max_rows_per_page index_fillfactor index_reservepagegap
 -------------------- ------------------------------------------------------------
       ------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
----------------------
       ------------------------ ---------------- --------------------
 index01              clustered, unique located on default
         col1, col2

                                0                0                    0

(1 row affected)
No defined keys for this object.
Object is not partitioned.
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with allpages lock
scheme.
The attribute 'concurrency_opt_threshold' is not applicable to tables with
allpages lock scheme.

 exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ----------------- ------------
            0              0          0                 0            0

(1 row affected)
concurrency_opt_threshold
 -------------------------
                         0
(return status = 0)
```

Figure 11. Screen of sp_help table01

This output gives us the following information:

- Table name
- Column name
- Data type, length, precision if applies
- Which segment it resides on, if any
- If the column allows nulls
- If any rules apply to this column
- If this is an `IDENTITY` column
- Primary key for the table
- Indexes associated with this table
- Type of index for indexes
- Segment for indexes

Below are several other stored procedures we will find useful during this conversion:

- `sp_helpcache` will give information about buffer cache allocation
- `sp_configure` will display configuration parameters when no parameters supplied
- `sp_helpconstraint` shows any constraints for objects
- `sp_helpdb` gives information about databases
- `sp_helpdevice` list device information
- `sp_helpgroup` shows the login/security groups set up in the database
- `sp_helpindex` lists index information
- `sp_helpkey` displays information about keys and tables
- `sp_helplanguage` shows language configures for this database
- `sp_helplog` displays information about the database logs
- `sp_helpobjectdef` displays object owner and type
- `sp_helprotect` is used to display permissions for an object
- `sp_helpsegment` lists the segments defined to a database
- `sp_helpsort` displays sort configuration
- `sp_helptext` will display the source for stored procedures
- `sp_helpuser` will list names assigned group for database users

5.1.1 Manual conversion

Now that we have a method to collect the necessary information, we will look at the resources available to do a manual conversion. The two manual methods we will address are these:

- Using Sybase SQL commands from the ISQL utility and the DEFNCOPY utility
- Generating DDLs using the Sybase Central product.

Both methods will require manually creating the DB2 UDB database, table spaces and all database objects, as well as editing all objects to conform to the DB2 UDB environment.

5.1.1.1 Command line method

To determine the type of objects in our Sybase database, we can execute a `select` statement as shown in Figure 12 to determine what type of objects we must convert.

```
1> use test1
2> go
1> select type, count(type) from sysobjects group by type order by type
2> go
 type
 ---- -----------
 P            115
 S             26
 TR            16
 U             35
 V             30

(5 rows affected)
1>
```

Figure 12. Sybase objects by type

Based on our output on the screen above, we can interpret as follows:

- P — We have 115 stored procedures.
- S — There are 26 system tables which we will not need to consider.
- TR — We have 16 triggers.
- U — There are 35 user tables.
- V — We have 30 views.

There are 35 user tables in the database and we will focus on those first. We first need to get this metadata into operating system files, so we can edit the CREATE TABLE DDL and change to the DB2 UDB format. One way to do this is to manually copy this data from an output screen from the sp_help stored procedure. This would be slow and tedious, so it is the least desirable option.

We can also use the DEFNCOPY utility to generate DDLs for views, rules, defaults, triggers, or stored procedures from a database, and edit them to be executed for DB2. Note that the DEFNCOPY utility cannot be used for tables.

We can execute the DEFNCOPY utility for each database object one-by-one, or we can compose a select command to create a script executing the DEFNCOPY utility.

Figure 13 shows the select statement that we used to generate defncopy statements for the views.

```
use test1
go
select 'defncopy -Usa -Sununbium -P out '+name+'.ddl dtp1  dbo.'+name
               from sysobjects
              where type = 'V'
              order by name
go
```

Figure 13. SQL to create output file with view names for DEFNCOPY

We can run the SQL above and create the actual output file with the defncopy statements. The generated script file would appear as shown in Figure 14.

```
defncopy -Usa -Sununbium -P out view01.ddl test1  dbo.view01
defncopy -Usa -Sununbium -P out view02.ddl test1  dbo.view02
defncopy -Usa -Sununbium -P out view03.ddl test1  dbo.view03
defncopy -Usa -Sununbium -P out view04.ddl test1  dbo.view04
defncopy -Usa -Sununbium -P out view05.ddl test1  dbo.view05
defncopy -Usa -Sununbium -P out view06.ddl test1  dbo.view06
defncopy -Usa -Sununbium -P out view07.ddl test1  dbo.view07
defncopy -Usa -Sununbium -P out view08.ddl test1  dbo.view08
defncopy -Usa -Sununbium -P out view09.ddl test1  dbo.view09
defncopy -Usa -Sununbium -P out view10.ddl test1  dbo.view10
defncopy -Usa -Sununbium -P out view11.ddl test1  dbo.view11
defncopy -Usa -Sununbium -P out view12.ddl test1  dbo.view12
defncopy -Usa -Sununbium -P out view13.ddl test1  dbo.view13
defncopy -Usa -Sununbium -P out view14.ddl test1  dbo.view14
```

Figure 14. Output file defncopy.views

After the job is run, we looked at the file created using a text editor such as the vi editor and then changed the file to add execute permissions using the `chmod +x` command so we could execute the file.

Once the DDL commands have been created, we must go through each file and look for incompatibilities and change to conform to DB2 requirements.

The processes above will need to be repeated for the stored procedures, defaults, triggers, and rules.

Since the DEFNCOPY utility does not generate DDL for tables, you will have to address tables either manually from the output of the `sp_help`, or generate DDLs using Sybase Central GUI.

It would make the process simpler if the security policy — who will have access to what — were to be defined before creating DDLs for the target DB2 database is complete, so we could place `GRANT` commands in our files with the proper objects.

When the DDL for each object is ready to be executed for the DB2 database, we can use the files as input to the DB2 command line processor (CLP) as follows:

```
db2 connect to db2_db
db2 -tvf filename.ddl -r output.txt -s -l errorlog.txt
```

Where:

- `db2_db` is the target db2 database.
- `filename.ddl` is your edited DDL file.
- `output.txt` is a file for output.
- `errorlog.txt` is a file for any errors generated.

5.1.1.2 Sybase Central

Sybase Central is the Sybase GUI interface product used to manage the server, and to create and maintain tables and other objects. It can also be used to collect the information above and create files containing the DDL. This can be done from Sybase Central as follows:

1. Click the **Databases** folder.
2. Click the database you want to convert.
3. Click the **User Tables** folder.
4. Right-click the table you want to generate the DDL.

5. Select **Generate DDL**, as shown in Figure 15. A separate window will pop up with the source for that table.

You can then save this to a file, and edit it with an editor such as the vi editor.

The DDL generated in this method will contain everything related to the table, such as permissions, constraints, and indexes.

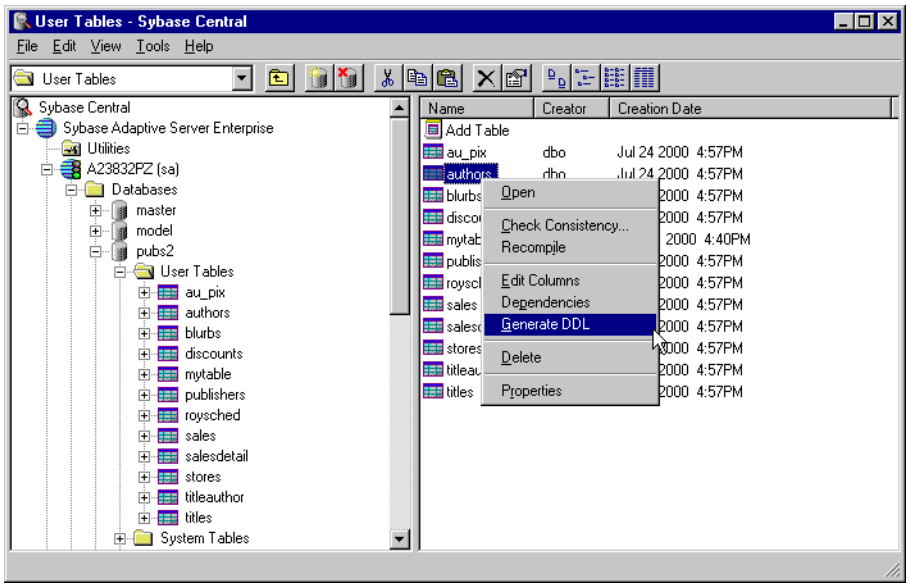

Figure 15. Sybase Central screen display

5.1.2 Using a conversion tool

Another option for the porting is the use of conversion tools. We have selected ManTech SQL Conversion Workbench for this project. The focus is not on the product itself, but that it is an option. More details are provided in A.1, “SQL Conversion Workbench” on page 229.

We will skip the installation process and go straight into the conversion process, using ManTech SQL Conversion Workbench. This process consists of four major steps:

- Unload metadata
- Load repository
- Build DDL
- Convert stored procedures

5.1.2.1 Unload metadata

To start the unload process, make sure you have already finished the **First Steps** process from the SQL Conversion Workbench icon. Then we will select the **SQL Server Unload Option** to begin this step.

The tool will unload the metadata of database objects, display a list of errors and warnings, and then wait for you to do the **Load Repository** function. When the unload completes, you will get a summary screen with statistics.

We are now ready to load the repository.

5.1.2.2 Load repository

This step simply loads the metadata created from the unload step into the repository database. It also creates a screen of errors and warnings.

5.1.2.3 Build DDL

After the repository is loaded with the metadata, you will have the opportunity to edit any members, and when satisfied, you can the generate the DDL for DB2 UDB.

The build DDL feature creates statements in DB2 UDB format to:

- CREATE DATABASE
- CREATE TABLESPACE
- CREATE TABLE
- CREATE VIEW
- CREATE TYPE

These DDL statements will need to be reviewed carefully to ensure that the file paths for the containers are where you want to place the tablespaces, to verify whether you want to use SMS or DMS storage and also for the placement of tables in table spaces. You may also want to create additional table spaces to place indexes or place certain tables in a table space by themselves.

When you are satisfied with the DDLs, you can use the output file as input into the DB2 command line processor (CLP) to build the objects in the DB2 database as follows:

```
db2 connect to db2_db
db2 -tvf filename.ddl -r output.txt -s -l errorlog.txt
```

Where:

- db2_db is the target db2 database.
- filename.ddl is your edited DDL file.
- output.txt is a file for output.
- errorlog.txt is a file for any errors generated.

When this step is completed, you can begin to convert the stored procedures.

5.1.2.4 Convert stored procedures

This tool will convert the Sybase stored procedures to the DB2 UDB requirements but that will be discussed in detail in a later chapter.

Depending on which conversion tool you have chosen, the steps to convert your Sybase database into DB2 database will be different from the ones we show in this section; however, we found that using the conversion tool had been the great contributing factor to reduce our workload on the project. We strongly recommend using such a conversion tool that assists you in the migration projects from Sybase to DB2 UDB. Please check the following Web site for the latest information on the migration tool from Sybase to DB2 UDB:

```
http://www-4.ibm.com/software/data/db2/migration/
```

5.2 Create DB2 instance

The first thing you will need to do is create your DB2 UDB instance.

Before creating an instance, you will need to create two UNIX groups and users for which you will need UNIX root authority, or have your Systems Administrator perform this function. These users will be the instance owner and the user under which fenced user defined functions and fenced stored procedures. These run in the different process from the database manager operating environment's process. Fenced user defined functions and stored procedures are documented in detail in the *Application Development Guide*, SC09-2949.

The creation of the DB2 instance is accomplished using the db2icrt command, or using the db2setup script on the install CD-ROM.

The following command creates a new instance db2inst1. You should execute this command after logging in as the root user:

```
/usr/lpp/db2_07_01/instance/db2icrt -u db2fenc1 db2inst1
```

Where db2fenc1 is the user that runs fenced user defined functions and stored procedures, and db2inst1 is the instance name.

The created instances can be listed by the `root` user using the following command:

```
/usr/lpp/db2_07_01/instance/db2ilist
```

Once you have created the new instance successfully, login to the server machine as the instance owner user ID, and then start the instance using the `db2start` command.

5.3 Create DB2 database

Now we can look at creation of our database. To create the database you will need to know the sort sequence you are currently using in the Sybase server.

5.3.1 Obtain sort sequence information from Sybase

With Sybase, your sort sequence is specified at the server configuration level whereas in DB2 it is at the database level. As shown in Figure 16, you can get the sort order for Sybase server using the `sp_helpsort` stored procedure.

```
1> sp_helpsort
2> go

 Collation Name                 Collation ID
 ------------------------------ ------------

 Loadable Sort Table Name
 ------------------------------

Sort Order Description

 ----------------------------------------------------------------------
 Character Set = 1, iso_1
     ISO 8859-1 (Latin-1) - Western European 8-bit character set.
 Sort Order = 50, bin_iso_1
      Binary sort order for the ISO 8859/1 character set (iso_1).
Characters, in Order


 ----------------------------------------------------------------------
     ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
   @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
   ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
   ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À
   Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à
   á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

(return status = 0)
```

Figure 16. sp_helpsort output

In the middle of this screen, we see that `Sort Order = 50, bin_iso_1`. This and the next statement tell us that we are using the binary sort option.

For DB2 the sort order is specified in the `CREATE DATABASE` command by way of the `COLLATE USING SYSTEM`, or `COLLATE USING IDENTITY` parameters.

Specifying `SYSTEM` will cause DB2 system to pick up a code page based on the `TERRITORY` value in the `CREATE DATABASE` command. The option `COLLATE USING IDENTITY` will do a binary sort.

Since our Sybase server was configured to use the binary sort, we need to specify `COLLATE USING IDENTITY` when we create our database. The only time you would need to use anything other than `IDENTITY` is if you have some non-standard sorting specified for Sybase.

5.3.2 Create database command for DB2

The `CREATE DATABASE` command we created follows:

```
CREATE DATABASE test1 ON /db2udb COLLATE USING IDENTITY
   CATALOG TABLESPACE
      MANAGED BY SYSTEM USING ( 'SYSCATSPACE")
      EXTENTSIZE 16 PREFETCHSIZE 16

   USER TABLESPACE
      MANAGED BY SYSTEM USING ( 'USERSPACE1')
      EXTENTSIZE 16 PREFETCHSIZE 32

   TEMPORARY TABLESPACE
      MANAGED BY SYSTEM USING ( 'TEMPSPACE1')
      EXTENTSIZE 16 PREFETCHSIZE 32;
```

The DDL above will create a database named `test1`, and we have specified the `COLLATE USING IDENTITY` which will give us a sorting sequence based on binary content of the columns. The option `IDENTITY` was chosen because our Sybase server's sort option was binary. No special code set is specified in the `CREATE DATABASE` command, and the locale of the current operating environment will determine the code set. In our case, the code page 819 (en_US) will be used.

We also told DB2 that we want to create the catalog table space `SYSCATSPACE`, user table space `USERSPACE1`, and temporary tablespace `TEMPSPACE1` as system managed space (SMS) table spaces, so we specified `MANAGED BY SYSTEM`.

Later, we will create user temporary table spaces, as well as table spaces into which we can place our table data.

The database is now created, and we are ready to set up the transaction log path and create some table spaces for our data.

5.3.3 Set up transaction log path

When a database is created, the log files will be created in the directory specified in the `ON` option of the `CREATE DATABASE` command. You will probably want to change this to separate the log files from the table spaces, both for performance and recovery reasons. This can be done by updating the configuration file as in the following example:

```
UPDATE DB CFG FOR test1 USING NEWLOGPATH '/path'
```

The new log file path for the `test1` database will become effective the next time the database is stopped and restarted.

You can also change the number of log files and size of the log files in this manner by changing the `LOGPRIMARY`, `LOGSECOND` or `LOGFILSIZ` values.

5.4 Create table spaces

Table spaces are the logical layer between the database and the table data stored in the database. As we discussed previously, there are two type of table spaces, System Managed Space (SMS) table spaces and Database Managed Space (DMS) table spaces. You will need to determine which types of table spaces to create. There can be both SMS and DMS table spaces within a database.

5.4.1 Designing table spaces

You will want to take some time and look at the current Sybase environment and decide how many table spaces you will need, what types of table spaces to create, and where you will place them. If your Sybase database is segmented, you will want to use the `sp_helpdb` and `sp_helpsegment` stored procedures to know the segments that your Sybase database uses and consider them in your design of the target DB2 database. The `sp_helpsegment segment_name` will show you which database objects are on the segment and which Sybase devices are in the segment.

Careful design of your table spaces can have a huge impact on the performance and recoverability you attain in the DB2 environment. Look at the tables and place any high activity tables in a table space by themselves if you have the resources to do so.

You may want to make sure you separate the indexes and the data. To do so, you will need to create two DMS table spaces, one to contain the data and the other to contain the index. This can be accomplished by specifying `INDEX IN tablespace_name` in your `CREATE TABLE` DDL. To separate the index and data, you must use DMS table spaces.

Remember also in the DB2 environment that the database backup and recovery is at the table space level. Table spaces can be backed up and recovered independently from the remainder of the database. This can be a big advantage if you have large tables.

You will also need to determine whether to use SMS or DMS table spaces, or a combination of the two. Here are some considerations:

- If you create an SMS table space, data pages for a table in the table spaces will be allocated as the table grows. Once an SMS table space is created, you cannot add new containers to the table space but the physical files in the existing containers can grow as long as the operating system's limitation. For the system catalog table space and temporary table spaces, choosing SMS table spaces is recommended. For details, see "Designing and Choosing Table Spaces" in Chapter 8, "Physical Database Design" in the *Administration Guide*, SC09-2944.
- When you create a DMS table space, you need to specify the container size, and you can add new containers later if you need to increase the table space size. You must use DMS table spaces to be able to place data and indexes in separate table spaces. Also, the DMS table spaces are more efficient.

To determine how large to make the table spaces, we can use the output from the `sp_helpdb db_name` stored procedure to get the size of the database. To see more details on how the space is allocated, we can use the `sp_spaceused` stored procedure. If executed without parameters, this shows the entire database, or you can specify a table name to see details on a specific table. Figure 17 shows a breakdown of space allocation, unused space, and space used by data and indexes for the table.

```
1> sp_spaceused
2> go
 database_name                 database_size
 ----------------------------- -------------
 test1                          17.0 MB
 reserved        data           index_size      unused
 --------------- -------------- --------------- ---------------
 7694 KB         4870 KB        752 KB          2072 KB
(return status = 0)
1> sp_spaceused table01
2> go
 name                rowtotal    reserved        data
        index_size      unused
 ------------------- ----------- --------------- ---------------
       --------------- ---------------
 table01 1           64 KB           2 KB
        2 KB            60 KB

(1 row affected)
(return status = 0)
```

Figure 17. sp_spaceused stored procedure

The first `sp_spaceused` stored procedure is without a parameter and shows us the information for the database.

The second `sp_spaceused` stored procedure is for the table `table01`. It shows that we have 1 row of data, and there is 64 KB reserved. The data and index are each using 2 KB and there is 60 KB unused. Collecting this data for data and indexes will help us to determine how many table spaces we need, and how large to make each.

Another important piece of data we will need to consider before creating our table spaces is how large to make our `PAGESIZE` parameter. The `PAGESIZE` default is 4 K bytes, but can also be specified as 8 K, 16 K, or 32 K. The page size will determine the length of rows that can be stored in the table and the maximum number of columns (see Table 13).

Table 13. PAGESIZE and maximum table size, # columns and row size

PAGESIZE	MAXIMUM TABLE SIZE	MAXIMUM # OF COLUMNS	MAXIMUM ROWSIZE
4 K	64 GB	500	4005
8 K	128 GB	1012	8101
16 K	256 GB	1012	16293
32 K	512 GB	1012	32677

See the *DB2 UDB Administration Guide: Implementation,* SC09-2944, for other considerations such as buffer pools.

5.4.2 Create tablespace statement

The prerequisites for creation of table spaces are the `db2icrt` command to create the instance, and the `CREATE DATABASE` command to create the database. This is different from Sybase, where you configure your data server, do `DISK INIT` to ready the devices, and then do `CREATE DATABASE` for database creation.

The format for the `CREATE TABLESPACE` for our database using SMS table space would look as shown in Figure 18:

```
CREATE REGULAR TABLESPACE tablespace1
    MANAGED BY SYSTEM
USING ( '/Dir1/db2inst2/smscont1' ,
        '/Dir2/db2inst2/smscont2')
EXTENTSIZE 16
PREFETCHSIZE 32 ;
```

Figure 18. DB2 create table space DDL

The options and values we chose were:

- `CREATE` — The DDL command.
- `REGULAR` — The type of table space. The options are `REGULAR`, `LONG`, `SYSTEM TEMPORARY`, or `USER TEMPORARY`.
- `TABLESPACE` — To tell DB2 what to create.
- `USING ( '/Dir1/db2inst2/oontainer1','/Dir2/db2inst2/container2')` — The container definitions. The physical files for the objects in the table space will be created under these directories.
- `MANAGED BY SYSTEM` — To tell DB2 we will use SMS for this table space.
- `EXTENTSIZE` 16 — To specify 16 pages to be written to a container before moving to the next.
- `PREFETCHSIZE` 32 — To specify number of pages to be read from the table space when data fetching is being performed.

The `EXTENTSIZE` and `PREFETCHSIZE` can also be specified as K(kilobytes), M(Megabytes), or G(gigabytes).

To create DMS table spaces, the format is very similar, as shown in Figure 19:

```
CREATE REGULAR  TABLESPACE tablespace2
    MANAGED BY DATABASE
    USING (  FILE '/Dir3/db2inst2/dmscont1' 5000,
           FILE 'Dir4/db2inst2/dmscont2' 5000)
    EXTENTSIZE 16
    PREFETCHSIZE 32;
```

Figure 19. Create DMS table space

The difference in format between SMS and DMS is in the `MANAGED BY` parameter, where we specified `DATABASE` and the `USING` parameter where we specified the `FILE` option. Also, we must specify the size in number of pages.

If you are planning to use DMS (device) table spaces on AIX operating system, you must first create new devices (logical volumes) for them as follows:

1. Log in as root.
2. Execute the `mklv` command to create logical volumes.
3. Execute the `chown` command to change the owner and group set of the logical volumes to match the DB2 instance owner and group.

We will also need to create user temporary space as discussed in 4.1.2.1, "DB2 table space" on page 32. This would be accomplished as shown in Figure 20:

```
CREATE USER TEMPORARY TABLESPACE user_tablespace
    MANAGED BY SYSTEM
    USING ( '/Dir5/db2inst2/smscont1',
        '/Dir6/db2inst2/smscont2')
    EXTENTSIZE 16
    PREFETCHSIZE 32;
```

Figure 20. Create user temporary table space

Note that `USER TEMPORARY` is specified in the `CREATE TABLESPACE` statement. When you create a declared temporary table using the `DECLARE GLOBAL TEMPORARY TABLE` statements, this temporary table space will be used.

5.5 Create user defined data types

In 4.2.5, “Other data types” on page 48, we discussed user defined data types. You will need to check the tables to see if you are using user defined data types.

In DB2 UDB, there are three user defined data types:

- User defined distinct type
- User defined structured type
- User defined reference type

Since Sybase supports only user defined types similar to the DB2 distinct type, we will focus on that. Consult the *DB2 UDB SQL Reference, Volume 1,* SC09-2974 for user defined structured data types and user defined reference data types.

5.5.1 Create data type statement

You can first execute a `SELECT` statement to extract the type name from the `systypes` table of the source Sybase database and look for user defined types by specifying `where usertype>99`. The select statement would look like this:

```
select name from systypes where usertype > 99
```

Earlier, when we checked the select from the `sysobjects` table within our sample database `test1`, there were no user defined data types. To show how you can derive the user defined types information from the Sybase source database, here we are using the `pubs2` database, which Sybase product provides. If you execute the select statement shown above for the `pubs2` database, the output would look like this:

```
1> use pubs2
2> go
1> select name from systypes where usertype>99
2> go
 name
 ------------------------------
 id
 mytype
 tid

(3 rows affected)
```

Then you can use the stored procedure sp_help to provide information about user defined types. The following is the information for the user defined data type id.

```
1> sp_help id
2> go
 Type_name       Storage_type    Length Prec Scale Nulls Default_name
         Rule_name       Identity
 --------------- --------------- ------ ---- ----- ----- ---------------
         --------------- --------
 id              varchar             11 NULL  NULL     0 NULL
         NULL                    0

(1 row affected)
(return status = 0)
```

From the output above, you can see the user define data type id has been created based on the data type VARCHAR and the length is 11. Thus, the following would create a user defined data type similar to the one in Sybase:

```
CREATE DISTINCT TYPE id AS varchar(11) WITH COMPARISONS
```

This data type can be referenced in the CREATE TABLE command as the data type for a column.

5.5.2 User defined data types and rules

In Sybase, you can create a rule using the CREATE RULE statement and bind it with a user defined type definition using the sp_bindrule stored procedure. You can also create a default using the CREATE DEFAULT statement and bind it with a user defined type definition using the sp_bindefault stored procedure. When you use a user defined data type bound with the rules or defaults in table column definitions, these columns inherit the rules, properties and defaults associated with that user defined type. This is a typical usage of user defined types in Sybase.

The usage of user defined types in DB2 is different from the one in Sybase. User defined types in DB2 are also based on the existing data types; however, you cannot add properties such as rules or defaults to user defined data types. All rules must be defined as constraints at the column level when you execute the CREATE TABLE or ALTER TABLE statements.

The purpose of using user defined types in DB2 is normally to benefit from strong typing (discussed in 4.2.5.2, “User defined data types” on page 50), which ensures that only those functions and operators explicitly defined on a

user defined type can be applied to that type of data and which avoids end-user mistakes during the assignment or comparison of different types.

For example, the user defined data type `id`, created in the previous section, cannot be compared with `VARCHAR` data directly. Sybase user defined types do not support strong typing and a user defined data type can be compared directly with its source data type.

Because the usage of user defined type is different between Sybase and DB2, if you have a user defined type bound with a rule and a default in Sybase, you should do either of the following when converting Sybase databases and applications to DB2:

- Create a user defined type in DB2 using the same name and the source data type.

  In this case, you can leave the `CREATE TABLE` DDL to use the user defined type but you need to add a constraint and specify the default value to the DDL to implement the functionality of the rule and the default object. Also you need to check your application code and put a cast function to all the SQL statements which compare the user defined type data to other data type.

- Do not create a user defined type in DB2; rather, use the source data type.

  In this case, you should modify the column definition of the `CREATE TABLE` DDLs to specify the source data type. You still need to add a constraint and specify the default value to the DDL to implement the functionality of the rule and the default object but you do not need to put a cast function to the SQL statements in your application code.

Checking all your application code and putting a cast function will be time consuming tasks. Unless you want to utilize strong typing, which Sybase does not support, we recommend using the system supported data types instead of user defined data types when you convert Sybase databases to DB2.

5.6 Creating tables

The `CREATE TABLE` commands for DB2 and Sybase are quite similar in their basic formats, but both have options the other does not have. We will focus primarily on the basics needed for our conversion. One of the differences we will likely encounter is that for Sybase, you create a table `ON` a segment, while for DB2, we create tables `IN` table spaces. Also pay special attention to the data types when defining the columns for the table. Differences for the DB2 `CREATE TABLE` command we should mention are:

- Sybase tables are created `ON segments`, DB2 tables are created `IN table spaces`.
- DB2 allows indexes and long data to be placed in different table spaces:
  - `INDEX IN tablespace_name` allows you to place the indexes in another table space.
  - `LONG IN tablespace_name` allows you to place long data types (LONG VARCHAR, LONG VARGRAPHIC and LOB) data types in another table space.
- For DB2 tables, all indexes for a table must be in the same table space, for example, data in `tablspc1` and all indexes in `tablspc2`.
- Sybase and DB2 support `CONSTRAINT`, which allows you to name the primary key as well as other unique constraints, but DB2 allows more options.
- Column default options are important to check for the following differences:
  - Sybase default values for columns is `NOT NULL`, DB2 is `NULL`.
  - Sybase allows you to specify a `DEFAULT` value for a column as well as specify a default value in User Data Types as mentioned above. Pay close attention to these.
- Sybase allows a `PCTFREE` specification on the `CREATE TABLE` command to specify the free space size on each data page. With DB2 there is a `PCTFREE` on the `CREATE INDEX` command, and also, you can alter a table to provide a `PCTFREE` value using the `ALTER TABLE` command.

5.6.1 CREATE TABLE statement

To illustrate the DB2 `CREATE TABLE` DDL needed to create a table from our database `test1`, first we should look at the table on the Sybase server. To look at the table we used `sp_help table01`, shown in Figure 21, to view the table characteristics.

We will need to collect information about the table as is exists on Sybase, so we can duplicate in DB2. Some things to look at are:

- Segments on Sybase
- Data types
- Primary key
- Foreign keys
- Constraints
- Indexes, both clustered and non-clustered

- IDENTITY columns — These require special consideration when specifying START WITH values, as discussed in 4.2.5.1, “Identity columns” on page 49.

```
1> sp_help table01
2> go
 Name                           Owner
         Type
 ------------------------------ ------------------------------
        ----------------------
 table01                        dbo
         user table

(1 row affected)
 Data_located_on_segment        When_created
 ------------------------------ --------------------------
 default                               Sep 19 2000  1:07PM
 Column_name     Type            Length Prec Scale Nulls Default_name
         Rule_name       Identity
 --------------- --------------- ------ ---- ----- ----- ---------------
        --------------- --------
 col1            varchar             40 NULL  NULL     0 NULL
         NULL                    0
 col2            varchar             10 NULL  NULL     0 NULL
         NULL                    0
 col3            float                4 NULL  NULL     0 NULL
         NULL                    0
 col4            smalldatetime        8 NULL  NULL     0 NULL
         NULL                    0
index_name           index_description
    index_keys
     index_max_rows_per_page index_fillfactor index_reservepagegap
 -------------------- ------------------------------------------------------------
     ------------------------------------------------------
     ----------------------- ---------------- --------------------
 table01_11040069641  clustered, unique located on default
    col1, col2
                           0                0                    0

(1 row affected)
No defined keys for this object.
Object is not partitioned.
Lock scheme Allpages
The attribute 'exp_row_size' is not applicable to tables with allpages lock
scheme.
The attribute 'concurrency_opt_threshold' is not applicable to tables with
allpages lock scheme.

 exp_row_size reservepagegap fillfactor max_rows_per_page identity_gap
 ------------ -------------- ---------- ------------------ ------------
            0              0          0                  0            0
 concurrency_opt_threshold
 -------------------------
                         0
(return status = 0)
```

Figure 21. sp_help table01 output

The first thing we want to see is if the table is created on a Sybase segment. We saw previously that this database did not use segments so that will not be a consideration here, although we might want to place the indexes in another table space.

Next we will look at data types. The review reveals one column `col4` with a definition of `SMALLDATETIME`. Previously we pointed out that DB2 did not support this data type, so we will have to resolve this. All other data types look good. To resolve this we will use the `TIMESTAMP` data type. This should be noted since it probably will require some program changes, and will need to be handled properly in the porting of the data.

Our DDL for `CREATE TABLE` looks like the following:

```
CREATE TABLE table01
(
   col1 VARCHAR(40) NOT NULL   ,
   col2 VARCHAR(10) NOT NULL   ,
   col3 FLOAT(8)    NOT NULL   ,
   col4 TIMESTAMP   NOT NULL   ,
CONSTRAINT index01 PRIMARY KEY ( col1,col2 )
)
IN TABLESPACE1;
```

The only data type we had to change in this example was the `SMALLDATETIME`, and we changed that to `TIMESTAMP`.

We also specified the `PRIMARY KEY` as defined in the Sybase table and created the table in the `TABLESPACE1` table space.

We should point out a potential problem during the conversion with duplicate table names. With Sybase the naming conventions are `dbname.owner.table_name`; and the DB2 UDB naming conventions are `schema.table_name`. If you have tables in your Sybase database with owners other that the database owner, you may possibly end up with duplicate table names on your conversion DDL for the new DB2 platform. You will need to check your Sybase database and resolve any potential duplicates before creating the DB2 UDB DDL. Figure 22 provides information about duplicates.

```
1> select name, count(name) from sysobjects
2>  group by name having count(name) > 1
3> go
 name
 ------------------------------ -----------
 table01                                  2

(1 row affected)
1>
```

Figure 22. Check for duplicate names in your Sybase database

5.6.2 Add constraints

Constraints are rules that the database manager enforces. We will discuss three types of constraints and their differences in implementation in DB2.

Both Sybase and DB2 support a unique constraint accomplished by creating an index and specifying `UNIQUE` in the `CREATE INDEX` DDL.

Both Sybase and DB2 have a `REFERENCES` option in the `CREATE TABLE` command to specify a column referenced by this column in another table. When specifying `REFERENCES`, that column must already exist in the other table.

Sybase has a `CREATE RULE` command, where a rule is created and then bound to a user data type or to a column using the `sp_bindrule` stored procedure. That rule is then bound to a column and those constraints will be checked on inserts or updates. A rule can specify limits, ranges or other data characteristics. They do not override column definitions.

Other than the rule, the constraints are very similar in format and function. You will need to look at each rule in the database and determine how it should be handled. Rules are stored in the `sysobjects` table in each database with a type of '`R`', so you can do the following to find your rules:

1. Execute `SELECT name FROM sysobjects WHERE type = 'R'` to find rules defined in the database
2. Use the `sp_helptext rule_name` stored procedure to get the information about rules

With Sybase, to find the constraints specified for a specific table, you can:

- Use the `sp_helpconstraint table_name` stored procedure get the information about constraints.

- Use the sp_help table_name stored procedure to get information on primary keys or foreign keys.
- You can also look at the index information to determine if the index has UNIQUE specified in the creation.

Following is an example of executing the sp_helptext stored procedure to get information about the rule pub_idrule in the pubs2 database:

```
1> sp_helptext pub_idrule
2> go
 # Lines of Text
 --------------
              1

(1 row affected)
 text

------------------------------------------------------------------------------
create rule pub_idrule
as @pub_id in ("1389", "0736", "0877", "1622", "1756")

(1 row affected)
```

DB2 allows you to create constraints when using the CREATE TABLE command, or add or alter constraints using the ALTER TABLE command. In the example below, we are going to ALTER the table PUBLISHER to allow only those rows where pubid is equal to 1389, 0736, 0877,1622, or 1756.

```
ALTER TABLE PUBLISHER
   ADD CONSTRAINT check_pubid
   CHECK (pubid IN ('1389', '0736', '0877', '1622', '1756')) ;
```

5.7 Create views

There are many good reasons to create views and use views to manipulate data:

- The base table can be changed to add new columns and the view will only be affected if the applications using the view require the use of the additional column(s).
- You can create views that return only subsets of data to the user. If you have multiple locations, this could be a way to allow users access to data for their location only.

- You can join tables for commonly accessed data, providing a more efficient retrieval of data.
- Views can be used to control access to sensitive data.
- You can sum values of a column, average the column values or select certain values such as maximum or minimum.

The database we have selected to convert has views for most of the tables. For the Sybase system, they are type 'V' in the system table `sysobjects`. When we did the select to determine the type of objects and how many objects there were earlier, we had 30 views created.

5.7.1 Create view statement

The `CREATE VIEW` DDL for Sybase and DB2 are very similar in the basic format. As with the `CREATE TABLE` command DB2 has many extensions which Sybase does not have. We will look at those necessary for the purpose of this conversion.

As shown in 5.1.1, "Manual conversion" on page 59, the `CREATE VIEW` DDLs for Sybase can be generated by the DEFNCOPY utility or the Sybase Central GUI. Here is a example of the `CREATE VIEW` DDL for Sybase:

```
CREATE VIEW view01 (col1, col2, col3, col4)
            as SELECT col1, col2, col3, col4 FROM table01
```

This is a fairly simple view, and you can use this DDL to create a view in DB2 as well.

5.7.2 Change the timestamp format using views

Another application of the view in our conversion effort is to return some of the values that are presented differently in an acceptable format for the application programs. For example, the `DATE` function can be used on a DB2 `TIMESTAMP` data type column to return the date in mm/dd/yyyy format from a `TIMESTAMP` column if the value of the Database country code is 001, which is USA. The Database country code is derived by the value of the `TERRITORY` parameter of the `CREATE DATABASE` command.

See the following `CREATE VIEW` DDL for DB2:

```
CREATE VIEW view02 (col1, col2, col3, col4)
            as SELECT col1, col2, col3, col4 FROM table02
```

Assuming the data type of the column `col4` is `DATETIME` in Sybase and converted to `TIMESTAMP` data type for DB2, if we leave the view as it is, the

format returned will be YYYY-MM-DD-HH.MM.SS.NNNNNN (Year-Month-Day-Hour.Minute.Second.Microseconds).

If we want to retrieve just the date portion of the column `col4`, we could create a view selecting `DATE(col4)` instead of `col4`. This will cause DB2 to return just the date portion of the `TIMESTAMP` in the MM/DD/YYYY format. This does not work for inserts, so we will need to change programs or SQL to provide the `TIMESTAMP` in the correct format. Here is the changed example:

```
CREATE VIEW view02 (col1, col2, col3, DATE(col4))
            as SELECT col1, col2, col3, col4 FROM table02
```

With this change, we now should see the date returned in MM/DD/YYYY format.

The primary precautions for views in conversion from Sybase to DB2 are these:

- Sybase does not allow deletes on multi-table views.
- For Sybase, inserts will not be successful unless all columns specifying `NOT NULL` are supplied for an insert.
- There are no options on the Sybase view not supported by DB2, although DB2 has many more options.

See the *DB2 UDB SQL Reference, Volume 1*, SC09-2974, and the *DB2 UDB SQL Reference Volume 2*, SC09-2975 for more details.

5.8 Create indexes

The `CREATE INDEX` command is also similar in format and function for the basic specifications. We will need to identify all the indexes on all tables as we discussed previously.

The unique index for Sybase is their method of enforcing the unique constraint. It is a must that we locate all the unique indexes in the Sybase database and create unique indexes in our DB2 database. You can find information using the `sp_help table_name` and `sp_helpindex table_name` stored procedures. These will provide the index name, type of index, and columns used for the index.

Non-unique indexes need to be identified also to ensure that we get optimum performance on our new DB2 database. There are also options in both Sybase and DB2 to create an index to allow backward scans or reads.

5.8.1 Indexes in Sybase and DB2

Both Sybase and DB2 `CREATE INDEX` commands function very similarly.

As with tables, Sybase indexes are created on segments, and DB2 indexes are created in table spaces. Both allow indexes and data to be in separate segments or table spaces, but Sybase allows multiple indexes to be on multiple segments. DB2 allows only one table space for all indexes.

Sybase naming structure for indexes is *table_name.index* and DB2 UDB is *schema.index.* This difference structure will cause potential problems during the conversion because there can possibly be duplicate names for the indexes created for the DB2 database. To illustrate we will look at two tables, `table01` and `table02`:

```
CREATE TABLE table01(col1 CHAR(2), c0l2 CHAR(4))
CREATE TABLE table02(col1 CHAR(2), c0l2 CHAR(4))
```

For Sybase, the following create index statements are valid.

```
CREATE UNIQUE NONCLUSTERED INDEX nc1 ON table01(col1)
CREATE UNIQUE NONCLUSTERED INDEX nc1 ON table02(col1)
```

The results will be that we have two indexes named `nc1` belonging to two different tables. To drop these indexes in Sybase, you would need to use the format:

```
DROP INDEX table01.nc1
```

Within DB2 indexes are contained in the catalog qualified by schema, so when these are converted, you will have both indexes in the same schema. You will need to go through your DDL and make sure you have no duplicate names for the indexes. One suggestion is to prefix all indexes with table name during the conversion process.

Another basic difference between the two is the creation and management of clustered indexes . When you create a table on the Sybase platform, then create a clustered index for that table, the data and index reside together on the same segment. For example, If you create a clustered index on the different segment from the one on which the base table was created, the table will be moved to the segment which you specify with the `CREATE CLUSTERED INDEX` command. This is a handy way to move a table to another segment.

DB2 maintains the data and index for a clustered index in *close proximity*, not actually with the data. Thus, if you have specified index and data in separate table spaces, even for clustered indexes they will be separate.

If you have created a clustered index on a table, DB2 maintains order in the data pages during data insert to the table when possible. To keep the order of the data pages, DB2 searches free space of the data page by the following algorithm:

1. Search target page. If not found, then:
2. Search any pages in same extent. If not found, then:
3. Append to end of table

5.8.2 CREATE INDEX statement

We will use the same table `table01` to illustrate the Sybase `CREATE INDEX` command:

```
CREATE UNIQUE CLUSTERED INDEX index01 ON table01(col4, col5)
```

We can now do the `sp_helpindex` stored procedure and look at our results.

```
1> sp_helpindex t_dtnd_node
2> go
 index_name           index_description
         index_keys
index_max_rows_per_page index_fillfactor index_reservepagegap
 -------------------- ------------------------------------------------------------
        ------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-----------------------
        ---------------------- ---------------- --------------------
 dtndcu               clustered, unique located on default
          node_nm, node_c, envt_ver_n
0               0                    0
(1 row affected)
(return status = 0)
1>
```

For the DB2 table we already have an index that was created when the table was created, so we will create an index on the columns col4 and col5 . Below is the format we used:

```
create index index01 on table01 (col4, col5)
   cluster allow reverse scans
```

The index we created for the table was on the column named `col4` and `col5` and it will allow reverse scans on that index.

For Sybase, the allow backward scan is a server-wide configuration option set on by the stored procedure as follows:

```
sp_configure 'allow backward scans', 1
```

For DB2, reverse scans are an individual index option on the `CREATE INDEX` command.

5.9 Database security

Security for the Sybase and DB2 systems are very similar to one another at the level of the `GRANT` and `REVOKE` functions. There is where any similarity stops. Sybase uses its own security, while DB2 uses the operating system for its security.

5.9.1 Sybase security

The security for the Sybase system consists of the `GRANT` and `REVOKE` commands for user access to user tables, columns within tables, stored procedures, and views. With these commands you can grant a user privileges to select, insert, or update tables or views. You can also grant a user permission to execute a stored procedure. These privileges can be removed using the `REVOKE` command. As mentioned previously, the resemblance ends here.

5.9.1.1 Sybase logins

Sybase security has a login for each user, much like an operating system. To allow a user access to the Sybase system, that user must be added to the Sybase server by using the stores procedure, `sp_addlogin`. The procedure requires a user name, password, and optionally can specify the default database for the user once logged in. The information is stored in the `syslogins` table in the `master` database.

At this point you will be logged into the server but cannot really do anything. You must add the user to each database within the server that the user is to have access to that database. This is done in a couple of ways:

- You can define user groups and group users by access requirements.
- You can alias one user to another user with like access requirements.
- You can add each user individually.

5.9.1.2 Sybase users

If you choose the first method of defining groups and assigning users to those groups, the first thing you need to do is add the group. This is done through the `sp_addgroup` stored procedures, as seen in the following example:

```
1> sp_addlogin user2, pass07, master
2> go
Password correctly set.
Account unlocked.
New login created.
(return status = 0)
1> sp_addgroup testgroup
2> go
New group added.
(return status = 0)
1> sp_adduser user2, user2, testgroup
2> go
New user added.
(return status = 0)
```

Using this method, the `GRANT` or `REVOKE` operations are done using the group name, and all users within that group have the same privileges. Using the example above, the grant privileges will be given to `testgroup`. The user is then added using the `sp_adduser` stored procedure and specifying the group parameter.

The second method requires setting up a user with the `sp_adduser` stored procedure, then `GRANT` or `REVOKE` privileges based on this user. Others with the same resource requirements would then be added using the `sp_addalias` stored procedure, as demonstrated here:

```
1> sp_addlogin user3, pass08
2> go
Password correctly set.
Account unlocked.
New login created.
(return status = 0)
1> sp_addalias user3, user2
2> go
Alias user added.
(return status = 0)
```

The user `user3` was added using `sp_addlogin`, then in the database we will use the `sp_addalias` to give user the same privileges as `user2`. Any changes to either the `testgroup` or `user2` will now be granted to `user3` or revoked from `user3`.

The last method is to add users using the `sp_adduser` stored procedure and then `GRANT` or `REVOKE` privileges individually.

```
> sp_addlogin user4, pass09
2> go
Password correctly set.
Account unlocked.
New login created.
(return status = 0)
1> sp_adduser user4, user4
2> go
New user added.
(return status = 0)
1> grant select on mytable to user4
2> go
```

Here we added the login, added the user to the database and granted select permission on the table `mytable`.

To determine the security on Sybase objects (tables, stored procedures, views), you use the `sp_helprotect` stored procedure, as shown in Figure 23.

```
1> sp_helprotect mytable
2> go
 grantor         grantee         type     action
         object          column     grantable
 --------------- --------------- -------- ----------------------------
        --------------- ---------- ---------
 dbo             testgroup       Grant    Delete
         mytable         All        FALSE
 dbo             testgroup       Grant    Insert
         mytable         All        FALSE
 dbo             testgroup       Grant    References
         mytable         All        FALSE
 dbo             testgroup       Grant    Select
         mytable         All        FALSE
 dbo             testgroup       Grant    Update
         mytable         All        FALSE
 dbo             user4           Grant    Select
         mytable         All        FALSE

(1 row affected)
(return status = 0)
1>
```

Figure 23. sp_helprotect stored procedure

The output from `sp_helprotect` tells you who granted the permissions, what privileges the user has, and whether the user can grant these privileges to another user, under the grantable column.

5.9.2 DB2 security

To protect data and resources associated with a database server, DB2 uses a combination of external security services and internal access control information. To access a database server you must pass some security checks before you are given access to database data or resources.

The first step in database security is called *authentication*, where the user must prove he/she is who he/she says he/she is. Authentication of a user is completed using the operating system or a separate product such as DCE. To authenticate a user, the security facility requires a user ID and a password. This user ID must have been created in the external security service. For example, if you use the UNIX operating system to perform the authentication, the user ID has to be created by the `mkuser` command and the password should be set by the `passwd` command.

The second step is called *authorization*, where the database manager decides if the validated user is allowed to perform the requested action or access the requested data. Authorization is performed using DB2 facilities. DB2 tables and configuration files are used to record the permissions associated with each authorization name. As in Sybase, the permissions to individual tables, columns, or other objects is done through the `GRANT` or `REVOKE` commands.

5.9.3 Migrating users and group definitions

Migrating user security information from Sybase to DB2 will require planning and gathering data on the current Sybase environment. For the login information to Sybase, there is no stored procedure to provide that information. It is very easy to construct a `SELECT` statement to extract the names:

```
select name from master..syslogins order by name
```

This statement provides a sorted list of the logins for the server.

Database users and security information

All user and group information can be extracted from the database as follows:

1. `sp_helpgroup` provides a list of the groups assigned in the database.
2. `sp_helpuser` provides a list of users, the group they are associated with and the login name if different from user name.
3. `sp_helprotect` extracts the table and other object permissions.

New DB2 users

You can use the information obtained from Sybase to set up the new DB2 users and groups. If you choose to use the operating system's security mechanism for the authentication, you should create users and groups for DB2 using the operating system's command such as `mkuser` and `mkgroup`.

Since DB2 users and groups are also users and groups created in the external security service, you should take care that the new users and groups will not interfere with the security policy in the server, particularly if the other application is running on the same server.

5.9.4 Granting authorities and privileges

As mentioned earlier, the `GRANT` and `REVOKE` commands to administer privileges is very similar between the two systems. There are some authorities and roles within Sybase that may need addressing depending on your usage of them.

Sybase authorities and roles

sa	This is the system adminitrator (SA) authority, which is the Sybase counterpart of the SYSADM authority in DB2. It allows all functions to be performed, including all administrative tasks. There are also ROLES that can be assigned to individual logins to allow a subset of the SA authorities.
sa_role	This role allows the SA to grant the authorities of SA to another user or users.
sso_role	This is the System Security Officer role and allows users with this role to administer security functions.
oper_role	This is the operations role and allows shutdown, backup, restore tasks for users with this role.
replication_role	This role is used only if you are using the replication server.
sybase_ts_role	This is Sybase Tech Support role allows the SA to perform special duties, or tasks requested by Sybase technical support.
user_defined roles	Users with this role can create roles and assign to users. In DB2, you can implement user defined roles using groups and granting specific priviledges to those groups.

DB2 authorities

DB2 has four levels of authority for support tasks:

SYSADM	Level for system administration tasks.
SYSCTRL	This provides ability to do almost any administrative task, but does not have authority to modify the instance configuration or access database objects unless granted specific permission.
SYSMAINT	Users with this authority can perform tasks like backups, update configuration files, restore table spaces, reorganize tables, or execute the `runstats` utility.
DBADM	Users with this authority can perform tasks like loading data tables, query tablespace states and other functions at the database level.

SYSADM, SYSCTRL, and SYSMAINT authorities are not assigned through the `GRANT` statement, but are set up or changed in the database manager configuration file. DBADM authority is assigned through the `GRANT` statement.

Privileges

Privileges in Sybase and DB2 are given to or removed from users or user groups by means of the `GRANT` and `REVOKE` commands. The formats are almost identical.

For DB2, the `GRANT` command format is identical to Sybase, as follows:

```
GRANT SELECT ON mytable TO user2
```

Chapter 6. Data conversion

This chapter deals with the movement of data from Sybase databases to DB2 UDB Version 7.1. The first part of the chapter discusses the two primary ways to unload data from Sybase. The second part of the chapter deals with the load of the data into DB2. The third part of the chapter explores the use of the DataJoiner to unload data from Sybase tables into integrated exchange format (IXF) files which contain table and index definitions, and use the IXF files to build the target tables with the indexes.

6.1 Unload data from Sybase

When unloading data from a Sybase table into a text file, you have a choice of using either the Sybase BCP (bulk copy procedure) tool or a select statement. The choice of which tool to use will depend primarily on the type of data that you will be moving over to DB2. You will need to consider the following when making that choice:

- The BCP utility is limited in the format of the data written out to the file.
- The BCP utility outputs a `datetime` or `smalldatetime` data type field into the format that DB2 IMPORT or LOAD utility does not accept.
- The BCP utility cannot put text or character fields in quotes to allow the same character as a row terminator (a carriage return is the default) to appear in the data.

These factors would suggest that the BCP utility only be used when moving tables in the conversion process consisting of basic data types such as character and integer. If the table contains a `datetime`, `smalldatetime` type fields then the best choice is to use the select statement to unload the data. The reason is that datetime fields in Sybase are converted during by the BCP utility. The result is that a date formerly represented as (12/15/99), for example, is now represented as (DEC 15 1999). This new format is not accepted by the DB2 IMPORT or LOAD utility, and will cause problems when you try and import it back into DB2 table.

The other case when you should use the select statement is when the table has text fields including carriage returns. In this case, you should use the select statement and put the text in character delimiters so that the DB2 IMPORT or LOAD utility can handle the carriage returns as a part of the text data.

In this section, first we discuss data unload using the BCP utility, and then show how to use the select statement.

6.1.1 Unload data using BCP utility

The BCP utility within Sybase has numerous options for unloading data from Sybase tables and databases. In this book, we will assume that since you are converting from Sybase to DB2, you have a basic understanding of the BCP process and its options and limitations. We will only discuss using the BCP process in two methods: without a format file, and with a format file. The basic command syntax for the `bcp` command is as follows:

```
bcp <database>.<owner>.<tablename> out <filename>.<filename extension>
<parameter options> -S<servername> -U<username> -P<password>
```

6.1.1.1 Simple BCP without a format file

Here is a simple example to use the BCP utility:

```
bcp test1.dbo.table01 out table01.bat -c -Sununbium -Usa -Padmin
```

By executing this command, you will obtain a text file `table1.bat` , including the unloaded data from the `dtp1.dbo.table01` table.

The basic example listed above consists of the following:

- `bcp`

  This is the primary Sybase command to initiate the BCP utility.

- `test1.dbo.table01`

  This is the database and table that we are unloading.

- `out`

  This is a primary Sybase command directing the output location.

- `table01.bat`

  This is the file that you are unloading the table into, you could further specify a directory for this file to reside in.

- `-c`

  This option specifies that the data be output in character format, and that the default field terminator (called the column delimiter in DB2) of tab “\t” is going to be used and new line “\n” will be used as the end of row delimiter. You can use the `-t` option and `-r` option to specify other characters for the field terminator and row terminator respectively.

- `-Sununbium`

  This specifies the server name.

- `-Usa`

  This specifies the user name.

- `-Padmin`

  This specifies the password.

6.1.1.2 Simple BCP process with a format file

If you do not want to unload all the columns or you want to change the order of the columns, you should use a format file with the BCP utility. Here is a simple example:

```
bcp test1.dbo.table02 out table02.bat -f "table02.fmt" -e
"bcperror02.txt" -Sununbium -Usa -Padmin -T 640000
```

Note that we specify the `-f` option with a format file instead of the `-c` option. In this example, the BCP utility will generate the output file in the format which is defined in the format file `table02.fmt`. The format file can be created interactively within the BCP process by just not specifying the `-f` or `-c` option, or if you created the format file in a previous BCP operation, it can be reused by specifying the file name. In our example, the source table's column definition is as follows:

```
create table table02 (
col01 varchar(10) not null,
col02 varchar(40) not null,
col03 varchar(40) ,
col04 varchar(10) ,
col05 varchar(10) ,
col06 varchar(50)
)
```

And the following is an example of the format file used with the BCP utility:

```
10.0
5
1  SYBCHAR  0  10 ", "  1     col01
2  SYBCHAR  0  40 ", "  2     col02
3  SYBCHAR  0  40 ", "  3     col03
4  SYBCHAR  0  10 ", "  4     col04
5  SYBCHAR  0  10 ", "  5     col05
```

This format file shows that all the columns except `col06` are to be unloaded, and that commas are used as the field terminators (column delimiters) and that (\n) is used as the end of line terminator.

Also, we specified the following options in our example. You can specify these options whether a format file is used with the BCP utility or not:

- `-e "bcperror02.txt"`

  This option adds a path to route any errors that occurred into an error file that we define.

- `-T 6400000`

  This option is used when the table you are unloading may contain a row that is more than 32 KB (16 pages) in size. If you have a row in the table that exceeds this size and you do not specify the `-T` option, then any data on that row past 32 KB will not be output. Therefore, you should review your data prior to the unload process to prevent any truncation of the row data.

6.1.2 Unload with a select statement

The preferred way to unload data from a Sybase table is to use the select statement. The select statement allows for maximum flexibility and control over how the data is going to be formatted and delimited. With a select statement you can unload the `datetime` and `smalldatatime` fields into the format which the DB2 IMPORT and LOAD utility will accept. You can also put double quotes or other characters around any text fields, which will eliminate any import problems that may occur because of special characters or carriage returns embedded in the fields. We will show two examples of using the select statement to unload data from Sybase. The first will be unloading the same table that we did with the BCP process described previously and the second select statement will be using a different table that has a `datetime` field.

6.1.2.1 Select statement unload process

The following is the select statement that would be used to unload the data from the same table for which we used the BCP utility in the previous examples. This example is saved to a file and then we would execute it within `isql` with a batch command.

```
set nocount on
go
USE test1
go
select '"' + col01 + '"' ,',',
       '"' + col02 + '"' ,',',
       '"' + ISNULL(col03, '')+ '"' ,',',
       '"' + ISNULL(col04, '')+ '"' ,',',
       '"' + ISNULL(col05, '')+ '"' ,',',
```

```
       '"' + ISNULL(col06, '')+ '"'
from dbo.table02
go
```

The first `'"' + col01 + '"'` extracts the `col01` column enclosed by double quote (") characters. When we import the unloaded file into DB2, the IMPORT or LOAD utility will handle the double quote characters as the character delimiters.

Note

Note that you should specify a character for the character delimiter that the unloaded data does not have. Otherwise, you cannot load the unloaded data into the target DB2 table correctly. It is the user's responsibility to ensure that the chosen delimiter character is not part of the data to be moved.

We then put a comma (,) that will be a column delimiter for the IMPORT or LOAD utility.

For the columns that may have NULL values, you should use the `ISNULL` function so that the select statement will not generate the string 'NULL' for null values.

Now that you have created the SQL file and saved it you need to execute it from the command line. You would enter the following statement:

```
isql -Usa -Sununbium -Padmin -otable02.dat -itable02.sql -n -w160 -h-1
```

This command consists of the following:

- `-Usa`

  This specifies the source database userID.

- `-Sununbium`

  This is the source server name

- `-Padmin`

  This is the userID's password

- `-otable02.dat`

  This is the output data filename that you want to route the output to.

- `-itable02.sql`

  This example uses `table2.sql` as the file name that contains your select statement.

- `-w160`

  This option specifies the line width of the output file. If the output row exceeds the default line width 80 bytes, you need to specify this option.

- `-h -1`

  This will specify that headers be included only once in the file.

6.1.2.2 Unloading data that contains text data type

When the source table has a text or image data type field which is longer than 32 KB, there is a consideration you should take. Here is a select statement for a table that contains a text data type column `col03`, and whose maximum length is 40 KB:

```
set textsize 40960
go
set nocount on
go
use dtp1
go
select '"' + col01 + '"' ,',',
       '"' + col02 + '"' ,',',
       '"' + col03 + '"'
from dbo.table03
go
```

Note that the `set textsize` statement is executed before the select statement. The `set textsize` statement specifies the maximum length, in bytes, of text or image data type field to be returned with a select statement. In our case we should specify 40960 as the maximum length of the text field. The default value is 32 K.

6.1.2.3 Unloading data with a datetime data type

The `datetime` and `smalldatetime` data types in Sybase present the biggest challenge when converting to DB2. The reason is that no matter how you enter a date into Sybase it will always be displayed with the character equivalent for the month. This creates a problem when trying to import that field into a DB2 `timestamp` field, because DB2 will not understand “JAN” or “FEB” as a month.

The table we are going to unload contains a `datetime` field whose first value is a date of “May 21 2000 1:15PM”. If we were to use the BCP utility with the `-c` option to unload this field, then it would be formatted as character and they would be mismatched when importing to DB2. However, by using a select statement with the `CONVERT` function, we can convert these fields over to a standard date time format to import into DB2. The following example will demonstrate how the `datetime` filed is converted. We have also included the Sybase table design as well as a sample line of data to help with understanding the SQL example.

Table design:

```
col01 varchar(10)   not null
col02 varchar(40)
col03 int           not null
col04 datetime      not null
```

Table data, first 2 rows:

```
data1,data2,1000,May 21 2000  1:15:53:716PM
data1,data2,1000,May 21 2000  1:15:53:913PM
```

Note that this is how the data would look if you were to use the BCP utility to unload the table.

Here is the select statement to unload and convert data:

```
set nocount on
go
USE dtp1
go
select '"' + col01 + '"' ,',',
       '"' + ISNULL(col02, '')+ '"' ,',',
       col03 ,',',
       `"' + substring(convert(char(12), col04,105),7,4)+
             substring(convert(char(12), col04,105),6,1)+
             substring(convert(char(12), col04,105),4,2)+
             substring(convert(char(12), col04,105),6,1)+
             substring(convert(char(12), col04,105),1,2)+
             substring(convert(char(12), col04,105),6,1)+
             substring(convert(char(12), col04,108),1,2)+
             substring(convert(char(12), col04,102),5,1)+
             substring(convert(char(12), col04,108),4,2)+
             substring(convert(char(12), col04,102),5,1)+
             substring(convert(char(12), col04,108),7,2)+
             substring(convert(char(12), col04,102),5,1)+
             substring(convert(char(26), col04,109),22,3) + `"'
```

```
from dbo.table04
go
```

You can break down this example as follows:

- `select '"' + col01 + '"',',','"' + ISNULL(col02, '')+ '"' ,',', col03`

  This part extracts the `col01` and `col02` column enclosed by double quote (") characters. For the column `col02`, you need to put the `ISNULL` function to avoid returning the character string `NULL` for null values. For the column col03, double quote characters are not necessary as it is an integer column.

- `'"' + substring(convert(char(12), col04,105),7,4)+...`

  The next 13 lines using the `SUBSTRING` function and `CONVERT` function extract the datetime data field and convert it into the format which the DB2 IMPORT or LOAD utility can accept. To perform the format conversion, we used the `CONVERT` function with the style parameter (the third argument) which determine the output format. Table 14 shows the style number we used in this example and the output format.

  In our example, the first 6 lines are for the date part (yyyy-mm-dd-), and the next 6 lines are for the time part (hh.mm.ss.). The last line is for the microsecond part.

Table 14. Style number and output format for the convert function

Style number	Output format
102	yyyy.mm.dd
105	dd-mm-yyyy
108	hh:mm:ss
109	hh:mi:ss:mmmAM (or PM)

You would save this select statement as the file `table04.sql` and execute it with `isql` using the following statement:

```
isql -Sununbium -Usa -Padmin -o"table04.dat" -i"table04.sql" -h-1
```

The result of this would be a file output to `table04.dat`. The first 2 rows of that file would look like this:

```
"data1","data2",1000,2000-05-21-13.15.53.716
"data1","data2",1000,2000-05-21-13.15.53.913
```

As you can see, we have converted the datetime fields over to a standard DB2 timestamp format.

6.1.3 Loading data into DB2 UDB

The process for loading the data into DB2 is handled using either the LOAD or the IMPORT utility. The differences between the two processes are quite dramatic when comparing performance. This stems from the fact that the IMPORT utility is executing SQL INSERTs, and DB2 needs to perform several functions such as checking constraints, firing triggers, and writing log records accordingly. The LOAD utility writes formatted pages directly into the database and performs faster than the IMPORT utility. Table 15 highlights the basic differences between the two processes.

Table 15. IMPORT utility and LOAD utility

IMPORT utility	LOAD utility
Slower than the LOAD utility when moving large amounts of data.	Fast for large amounts of data because it writes formatted data.
Will fire triggers.	Will not fire triggers.
Will check constraints.	Will not check constraints.
Creation of tables and indexes supported with PC/IXF format input file.	Tables must exist.
Supports import into tables and views.	Supports loading into tables only.
The table spaces in which the table and its indexes reside are online for the duration of the import.	The table spaces in which the table and its indexes reside are offline for the duration of the load operation.
All rows are logged .	Minimal logging is performed.

6.1.4 DB2 IMPORT and LOAD utilities

The DB2 IMPORT utility fits into the Sybase-to-DB2 conversion process in several ways. First, it can be used as a design and proof-of-concept utility in the early stages of the conversion effort. Second, it can play a primary role in moving small and medium size tables. As we will discuss later in this chapter, the IMPORT utility can also create tables and indexes if you have PC/IXF format data files. The IMPORT utility is a very versatile and flexible tool in the conversion process. For tables with large amount of data, you should use the LOAD utility because it is faster than the IMPORT utility.

The basic syntax to execute the IMPORT and LOAD utilities is as follows:

```
IMPORT FROM filename OF filetype MODIFIED BY filetype-mod MESSAGES
filename INSERT INTO table-name

LOAD FROM filename OF filetype MODIFIED BY filetype-mod MESSAGES
filename INSERT INTO table-name
```

As you can see, the basic syntax for these utilities is very similar. Both of them load the data from the file specified by the `FROM` parameter. If you specify the `REPLACE` option instead of the `INSERT` option, the target table will be emptied first, and then the data will be loaded. There are more options that can be specified but for the purposes here we will only be using options that pertain to the data we unloaded from Sybase. In addition we will also assume that the tables have already been created in the DB2 table space.

Other considerations involve the use of delimiters. The default column delimiter for Sybase BCP utility is a tab; however, for DB2 IMPORT and LOAD utilities, it is a comma. You may want to standardize the use of one of these to prevent errors when loading the data.

If you decide you would like a more detailed look at the IMPORT or LOAD utility, refer to the *DB2 UDB Data Movement Utilities Guide and Reference*, SC09-2955 for a more complete understanding of the utilities and the parameters available.

6.1.5 Load data to DB2 from Sybase BCP file

Make sure you have connected to the target database before performing these commands.

Here is an example of the basic `IMPORT` command using the `REPLACE` option:

```
IMPORT FROM table01.dat OF DEL MODIFIED BY COLDEL0x09 MESSAGES
table01.msg REPLACE INTO table01
```

If you want to use the LOAD utility, the command would be the following:

```
LOAD FROM table01.dat OF DEL MODIFIED BY COLDEL0x09 MESSAGES table01.msg
REPLACE INTO table01
```

We are using the file type modifier `COLDEL0x09` in this example. This means we are using the tab character as column delimiters. As we addressed in 6.1.1.1, “Simple BCP without a format file” on page 92, the tab character is the default column delimiter of the Sybase BCP utility. If you chose to use the default column delimiter when unloading the data using the BCP utility, `COLDEL0x09` must be specified.

If you explicitly specify the comma character as the column delimiter using the `-t` option of the BCP utility or using the format file as shown in 6.1.1.2, “Simple BCP process with a format file” on page 93, you do not need to specify the file type modifier `COLDEL` option for the IMPORT or LOAD utility since the comma character is the default column delimiter.

```
IMPORT FROM table01.dat OF DEL MESSAGES table01.msg REPLACE INTO table01
```

When you are loading the data into a remote database using the LOAD utility, you need to specify the `CLIENT` option. For example, when the unloaded data is on a Windows workstation and the target database is on a remote AIX server, the load command you should execute from the Windows workstation would be as follows:

```
LOAD CLIENT FROM C:\output\table01.dat OF DEL MODIFIED BY COLDEL0x09
MESSAGES table01.msg REPLACE INTO table01
```

Note that the input file should be specified using the absolute path.

When the `LOAD` command finishes executing, you can open the message file to view the results. Here are the contents of the message file associated with running the import command just shown:

```
SQL3109N  The utility is beginning to load data from file
"table01.dat".
SQL3110N  The utility has completed processing.  "5" rows were read from the input file
SQL3221W  ...Begin COMMIT WORK. Input Record Count = "5".
SQL3222W  ...COMMIT of any database changes was successful.
SQL3149N  "5" rows were processed from the input file.  "5" rows were
successfully inserted into the table.  "0" rows were rejected.
```

6.1.6 Load data to DB2 from Sybase select statement file

This example is going to load the data from the data file created in 6.1.2, “Unload with a select statement” on page 94. This file was output with double quotes (“) around all of the character fields, and these are the default character delimiters. If you have specified a character other than the double quote, you need to specify it using the file type modifier `CHARDEL` of the IMPORT/LOAD utility. Here is a sample `IMPORT` command:

```
IMPORT FROM table02.dat OF DEL MODIFIED BY CHARDEL” DELPRIORITYCHAR
MESSAGES table02.msg RESTARTCOUNT 2 REPLACE INTO table02
```

In this example, we also specified the file type modifier DELPRIORITYCHAR. This is necessary when the character fields have carriage returns as a part of the data. By default, the IMPORT or LOAD utility interprets a carriage return as the end of row. However, if you specify the file type modifier DELPRIORITYCHAR, carriage returns within text enclosed by the character delimiter will be handled as a part of the data.

Also note that we specify the RESTARTCOUNT 2 option to skip the first 2 rows of the input data because they are the header generated by the select statement and should not be loaded into the target DB2 table.

If you want to use the LOAD utility, the command would be:

```
LOAD FROM table02.dat OF DEL MODIFIED BY CHARDEL" DELPRIORITYCHAR
MESSAGES table02.msg RESTARTCOUNT 2 REPLACE INTO table02
```

6.1.7 Load data to DB2 from Sybase with identity columns

As discussed in 4.2.5.1, "Identity columns" on page 49, both Sybase and DB2 can have tables with an identity column, which generates a unique sequential number for each row in the table. When you are converting a Sybase table with an identity column to DB2 table, the unload process is same as the other tables; however, the IMPORT and the LOAD utility have some parameters how to deal with identity values, and you need to use one of them depending on your requirement.

When loading the data into a table with an identity column, you have two options:

- Have all the identity values regenerated by DB2 UDB when loading the data
- Retain the identity values stored in the source Sybase table

Next we will discuss which parameter of the LOAD or IMPORT utility you should use for each case.

Note

When you create a table with an identity column in a DB2 database, make sure to specify the GENERATE ALWAYS option in the identity column definition. This option will ensure that any part of your application using the identity column will retain the same functionality that had within Sybase.

6.1.7.1 Regenerate identity values

If you prefer to have all the identity values regenerated by DB2 UDB when loading the data, specify the file modifier `IDENTITYIGNORE` of the IMPORT or LOAD utility. This modifier instructs the LOAD or IMPORT utility that data for the identity column is present in the input data file but should be ignored. This results in all identity values being generated by the utility.

Here is a sample `IMPORT` command:

```
IMPORT FROM table05.dat OF DEL MODIFIED BY CHARDEL" DELPRIORITYCHAR
IDENTITYIGNORE MESSAGES table05.msg RESTARTCOUNT 2 REPLACE INTO table05
```

Here is a sample `LOAD` command:

```
LOAD FROM table05.dat OF DEL MODIFIED BY CHARDEL" DELPRIORITYCHAR
IDENTITYIGNORE MESSAGES table05.msg RESTARTCOUNT 2 REPLACE INTO table05
```

In these example, we are assuming that the file `table05.dat` is the unloaded data from Sybase using a select statement and includes the identity values.

6.1.7.2 Retain the identity values

When you are converting a Sybase table with an identity column to DB2 table, and you wish to retain the identity column value from the source Sybase table, use the file modifier `IDENTITYOVERRIDE` of the LOAD utility. This modifier instructs the LOAD utility to load the identity values supplied in the input file. This file modifier is not available for the IMPORT utility. Here is a sample `LOAD` command:

```
LOAD FROM table05.dat OF DEL MODIFIED BY CHARDEL" DELPRIORITYCHAR
IDENTITYOVERRIDE MESSAGES table05.msg RESTARTCOUNT 2 REPLACE INTO
table05
```

In this example, we are assuming that the file `table05.dat` is the unloaded data from Sybase using a select statement and includes the identity values.

Note

When you wish to retain the identity column value from the source Sybase table, specify the highest identity value in the source table plus 1 with the `start with` option in the identity column definition of the `create table` statement. Otherwise, after loading the data into the target DB2 table, a future insert may generate a duplicate identity value. This is because the LOAD utility does not keep track of the values when using the file modifier `identityoverride`. Therefore, the first insert after the loading will generate the identity value which is specified with the `start with` option of the `create table` statement.

6.2 Data conversion using DataJoiner

DataJoiner provides the ability to view popular relational data sources (DB2 Family, Oracle, Sybase, SQL Server, Teradata, and others) and non-relational data sources (IMS and VSAM) as if they were local DB2 data sources.

Using DataJoiner may represent a significant savings of time and effort if it is available to you. Once you have created nicknames referring to Sybase tables respectively, you can handle the Sybase tables as if they were DB2 tables. Thus, you can use the EXPORT utility to unload data from Sybase tables. The benefit of using the EXPORT utility is that you can unload data from Sybase tables into integrated exchange format (IXF) files.

An IXF file includes not only unloaded data, but also table structures such as column name, data type, and primary key name. If a source Sybase table has indexes, the exported IXF file from the table has definitions of those indexes as well. Once you have unloaded a Sybase table and indexes on the table into an IXF file, you can use it as the input for the IMPORT utility. Executing the IMPORT utility with the `CREATE` option against the target DB2 database, you can accomplish the tables, indexes, and data conversion all at one time.

Of course, before using DataJoiner, the database design will have to be implemented in DB2 on the server. That is, the same jobs that we used to create the database, table spaces, and so forth will have to be run to use this approach as well.

Note

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. Please refer to 'Recreating an Exported Table' in Chapter 2 'Import' of the *Data Movement Utilities Guide and Reference*, SC09-2955 to see which attributes of the original table are retained.

6.2.1 Conversion scenario

In our project, we have performed the following steps to convert table and indexes from the source Sybase database to the target DB2 database.

- Installing the DataJoiner product, creating a DataJoiner instance and database, and configuring DataJoiner to access Sybase tables

- Running the EXPORT utility by the DataJoiner instance owner and exporting source Sybase tables to IXF files
- Running the IMPORT utility with the `CREATE` option by the DB2 instance owner, and creating tables, indexes, and importing data at a time

Figure 24 shows the table and index conversion scenario using DataJoiner.

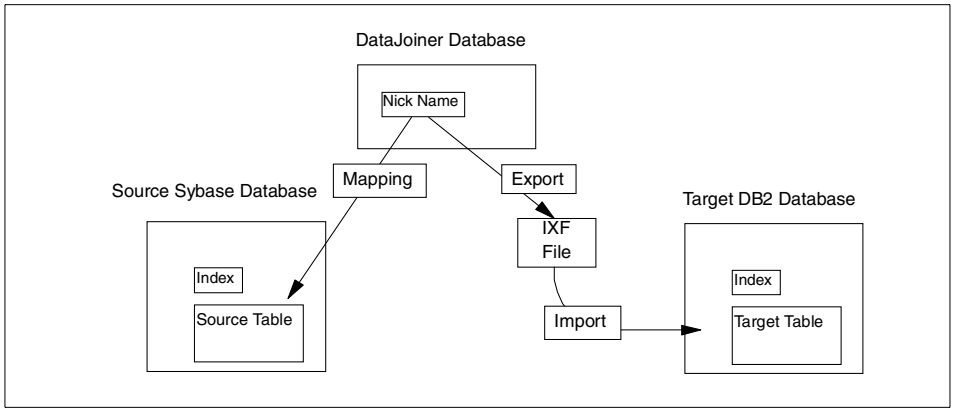

Figure 24. Table conversion using DataJoiner

6.2.2 Installing and configuring DataJoiner for AIX

In our project, we have installed DataJoiner in the same RS/6000 as we have installed DB2 UDB V7.1 and Sybase Adaptive Server V12. If you want to install the DataJoiner product on a different machine, you need to install Sybase Open Client with DataJoiner so that DataJoiner can access Sybase databases.

DataJoiner Version 2.1.1 on AIX is the version that we used in our project. For more detailed information on the installation and configuration of DataJoiner, refer to *DataJoiner Implementation and Usage Guide,* SG24-2566.

6.2.2.1 Installing and configuring the base product

After inserting the DataJoiner distribution media in the appropriate drive, you can install the product as follows:

1. From an AIX shell and with `root` user, run the command:

   ```
   smitty install_latest
   ```

2. Input your installation device identifier (for example, `/dev/cd0`).

3. On the next screen, select the components you want to install (with F4 key), or select `all_latest`

After installing DataJoiner base product, the configuration includes the following steps:

1. Create an AIX group for the DataJoiner instance (for example, `djinst`) using `smit` or with the AIX command:

   ```
   mkgroup -A djinst
   ```

2. Create an AIX user for the DataJoiner instance (for example, `djinst`), belonging to the `djinst` group previously created, using `smit`, or with the AIX command:

   ```
   mkuser pgrp=djinst djinst
   ```

3. Create a DataJoiner instance for the `djinst` user previously defined, with the following command:

   ```
   /usr/lpp/djx_02_01_01/instance/db2instance djinst
   ```

   The three commands previously described have to be issued by `root` user.

4. At this point, you can login to the AIX system as the `djinst` user and add the following line in the `.profile` file (running in a Korn shell):

   ```
   . /home/djinst/sqllib/db2profile
   ```

   Where `/home/djinst` is the `djinst` home directory.

5. Open the `db2profile` file in the `sqllib` subdirectory of `djinst` home directory (`/home/djinst`) and modify the lines as follows:

   ```
   DB2COMM=TCPIP
   export DB2COMM
   ...
   DJXCOMM='dblib'
   export DJXCOMM
   ```

   The environment variable `DJXCOMM` specifies the data access modules that DataJoiner loads when the DataJoiner instance is started. You need to specify a different data access module for each data source type.

   In our project, we used the DB-Library to access Sybase tables, therefore we specify `dblib` to the `DJXCOMM` environment variable here. You can also use the Client-Library by specifying `ctlib` to the `DJXCOMM` environment variable. The DB-Library and the Client-Library are contained in Sybase Open Client.

The DJXCOMM environment variable can have multiple values. If you also intend to access DRDA data sources (such as DB2 for OS/390) from DataJoiner, specify drda (communication protocol is APPC) or drdaIP (communication protocol is TCP/IP) for the DJXCOMM environment variable. Then DataJoiner will access DRDA data sources using the DRDA protocol through the DDCS component which is included in DataJoiner. To access tables on DB2 UDB V5 or V6 from DataJoiner, specify db2ra (DB2 private protocol), drda (DRDA APPC), or drdaIP (DRDA TCP/IP).

You cannot use db2ra for DB2 UDB V7. Using the DB2 private protocol, DB2 clients can only access DB2 servers with a release level which is in the range of one level lower than the client to two levels higher than the client. The Client Application Enabler (CAE) included in DataJoiner V2.1.1 is at the Version 2 level and thus cannot access DB2 UDB V7.1 server directly (you will receive SQLCODE -5048). You should specify drdaIP or drda for the DJXCOMM environment variable so that DataJoiner can communicate with DB2 UDB V7.1 using DRDA protocol.

6. Execute the djinst .profile file, for example, with the command:

```
. $HOME/.profile
```

7. At this point, the DataJoiner can be started by the djinst user with the command:

```
db2start
```

8. A DataJoiner database can be created, for example, with the command:

```
db2 create database DJDB
```

6.2.2.2 Configuring DataJoiner to access Sybase

In this section we describe the configuration of DataJoiner for the connection to the Sybase database.

Configuration of DataJoiner Data Access Modules

1. Login as *root* and add the following lines in the /.profile file:

```
SYBASE=/usr/sybase/OCS-12_0
PATH=$PATH:$SYBASE/bin
export SYBASE PATH
```

2. Then, execute the .profile file with the command:

```
. /.profile
```

3. Run the following command from the /usr/lpp/djx_02_01_01/lib directory to create the Data Access Module using Sybase DB-Library:

```
make -f djxlink.makefile dblib
```

You will see the following warning messages, which can be ignored:

```
[root@ununbium:/usr/lpp/djx_02_01_01/lib]make -f djxlink.makefile dblib
       ld -o dblib  -e _no_start  -bE:sybase.exp  -bI:libdb2e.exp  -bM:SRE  -K
 -L/usr/sybase/OCS-12_0/lib  -lc -lsybdb  libsybase.a  libdjexits.a
ld: 0711-327 WARNING: Entry point not found: _no_start
ld: 0711-319 WARNING: Exported symbol not defined: remoteLockEP
ld: 0711-319 WARNING: Exported symbol not defined: xaOpenEP
ld: 0711-319 WARNING: Exported symbol not defined: xaCloseEP
ld: 0711-319 WARNING: Exported symbol not defined: xaStartEP
ld: 0711-319 WARNING: Exported symbol not defined: xaEndEP
ld: 0711-319 WARNING: Exported symbol not defined: xaPrepareEP
ld: 0711-319 WARNING: Exported symbol not defined: xaCommitEP
ld: 0711-319 WARNING: Exported symbol not defined: xaRollbackEP
ld: 0711-319 WARNING: Exported symbol not defined: createServerEP
```

If you want to use the Client-Library, execute the following command:

```
make -f djxlink.makefile ctlib
```

Configuration of Sybase connection (DB-Library)

1. Login as `djinst` and add the following lines in his `.profile` file (`/home/djinst/.profile`):

   ```
   SYBASE=/usr/sybase
   SYBASE_OCS=OCS-12_0
   PATH=$PATH:$SYBASE/bin
   export SYBASE SYBASE_OCS PATH
   ```

2. Run this `.profile` file again, as follows:

   ```
   . /home/djinst/.profile
   ```

   Where `/home/djinst` is the `djinst` home directory.

Before configuring DataJoiner to connect to the Sybase databases, you need to put the `interfaces` file on `/home/djinst/sqllib` directory, where `/home/djinst` is the `djinst` home directory. The `interfaces` file entry define how Sybase servers and clients find and communicate with each other on the network.

If you have installed DataJoiner on your Sybase machine as our project scenario, just copy `/usr/sybase/interfaces` to `/home/djinst/sqllib` directory, or create a link by executing the following command:

```
ln -s /usr/sybase/interfaces /home/djinst/sqllib
```

If your DataJoiner system is different from the Sybase system, you have to configure the Sybase Open Client to reach the Sybase database from the DataJoiner system. What you need to do is add a server entry to the `/usr/sybase/interface` file for the Sybase Open Client on your DataJoiner system using the dsedit tool or a text editor as the following example:

```
ununbium
        master tcp ether ununbium 4100
        query tcp ether ununbium 4100
```

In the example above, `ununbium` is the host name of the Sybase server system, and `4100` is the port number used to communicate with the Sybase server. Once you have modified the interfaces file, copy it to the `/home/djinst/sqllib` directory, or create a link referring the `/usr/sybase/interfaces` file.

You can test the connection between the Sybase Open Client and the Sybase Adaptive Server by executing the following command:

```
isql -Sununbium -Usa -Ppswd
```

In the example above, `ununbium` is the server name, `sa` is the login name, and `pswd` is its password.

Refer to the Sybase documentation *Installation Guide Sybase Adaptive Server Enterprise,* Document ID:35892-01-1200-01, for the information on the Sybase Open Client configuration.

Configuration of DataJoiner mappings and nicknames

Now, you can create the *server mapping* for your Sybase database.

First you need to connect to the DataJoiner database (with `djinst` user, from an AIX shell), as follows:

```
db2 connect to djdb
```

Then, you run the `CREATE SERVER MAPPING` statement with the following command:

```
db2 create server mapping from <SERVER NAME> to node <NODE NAME> \
database <SOURCE NAME> type <DATABASE SERVER TYPE> \
version <VERSION NUMBER> protocol \"<DAM>\"
```

Where:

- SERVER NAME is a unique name of your choice (sybasedb, in our case).
- NODE NAME is the server name used in *interfaces* file (ununbium, in our case).
- SOURCE NAME is the source Sybase database name (test1, in our case).
- DATABASE SERVER TYPE is sybase.
- VERSION NUMBER is the version of your Sybase database (12.0, in our case).
- DAM is the data access module you are using (dblib, in our case).

In our environment, here is the command we used:

```
db2 create server mapping from sybasedb to node \"ununbium\" \
  database \"test1\" type sybase version 12.0 protocol \"dblib\"
```

Note that the node name, database name, data access module name are enclosed in double quotes because they are required to preserve case-sensitivity.

With the following command, create the user mapping for the Sybase connection:

```
db2 create user mapping from djinst to server sybasedb \
  authid user01 password user01
```

Where:

- djinst is the local user.
- sybasedb is the server mapping previously created.
- user01 is the source Sybase database user.
- user01 is his password.

At the end, a local nickname for a remote Sybase table can be created with the following command:

```
create nickname <NICKNAME> for <SERVERNAME.REMOTEUSER.TABLE>
```

In our case, this command would be:

```
create nickname table1 for sybasedb.dbo.table1
```

You can test the connection by selecting from the table1 nickname, as follows:

```
db2 "select * from table1"
```

Column and index length limits

When a nickname is created for a table, DataJoiner stores the names of the table's or view's columns in the catalog. Since DataJoiner databases are at the version 2.1 level, the maximum allowable length of column names is 18 characters. If a nickname is being created for a Sybase table that has columns whose name exceeds this length, DataJoiner truncates them to 18 characters.

The maximum allowable length of DB2 index names is also 18 characters. If a nickname is being created for a table that has an index whose name exceeds this length, the entire name is not cataloged. Rather, DB2 truncates it to 18 characters.

If the truncated version of column name is not unique within the table, or if the truncated version of the index name is not unique among the other names of indexes that belong to the same table, DB2 UDB will attempt to make it unique by a further modification. See the `CREATE NICKNAME` statement's page of the *DB2 UDB SQL Reference, Volume 2,* SC09-2975 for detailed information on how this is done.

If you export data into an IXF file using a nickname that has truncated columns, preserved column names in the IXF file will be also truncated. Therefore, the target table created from the IXF file using the `CREATE` option of the IMPORT utility will have truncated column names. If a source Sybase table has columns whose name length is greater than 18 characters, you should create the target table in the DB2 database manually to preserve the original column names in the source tables, and then import data using the `INSERT` option of the IMPORT utility (or using the LOAD utility).

The same is true for index names. If a source Sybase table has an index whose name length is greater than 18 characters and you want to preserve the index name in the target DB2 database, you should create the target table and index in the DB2 database manually, and then import data using the `INSERT` option of the IMPORT utility (or using the LOAD utility).

6.2.3 Exporting tables using DataJoiner

As we explained in 6.2.2, "Installing and configuring DataJoiner for AIX" on page 105, in our DataJoiner database we have created a set of nicknames for the source Sybase tables. You are now ready to export Sybase tables into IXF files using the EXPORT utility.

First you need to connect to the DataJoiner database (with `djinst` user, from an AIX shell), as follows:

```
db2 connect to djdb
```

Then execute the following command for each nickname:

```
db2 export to <output file> of ixf messages <message file> \
  select * from <NICKNAME>
```

In our case, for example, this would be:

```
db2 export to table1.dat of ixf messages table1.msg \
  select * from table1
```

By executing the EXPORT utility for each nickname, you can obtain a set of IXF files including unloaded data, table definitions, and index definitions.

The IXF files include not only unloaded data, but also table structures such as column name, data type, and primary key name. If a source Sybase table has indexes, the exported IXF file from the table has the definition of those indexes as well.

Note

The EXPORT utility truncates the LONG field to 32700 bytes.

6.2.4 Importing tables from IXF files

Once you have exported all Sybase source tables into IXF files, you can create tables in the target DB2 database and load data using the Import utility.

First you need to connect to the target DB2 database (with `db2inst1` user in our case), as follows:

```
db2 connect to test1
```

You should execute the following command for each exported IXF file:

```
db2 import from <exported IXF file> of ixf messages <message file> \
   create into <target table>
```

In our case, for example, this would be:

```
db2 import from table1.dat of ixf messages table1.msg \
   create into table1
```

Note

As we discussed in 5.8, “Create indexes” on page 81, a Sybase table may have the same name indexes as the other tables, whereas each index name is unique within a DB2 database. Therefore, when you use the Import utility to create a DB2 table, you may receive error messages such as the following, and cannot convert indexes:

```
SQL0601N The name of the object to be created is identical to the
existing name "DJINST.INDEX1" of type "INDEX". SQLSTATE=-42710

SQL3189N The previous message refers to index "DJINST.INDEX1" with
columns "+COLA+COLB+COLC".
```

The error SQL0601N indicates that the target database already has the index named `DJINST.INDEX1` and DB2 does not allow to use the same name. In this case, you need to create the index manually using a different name. The message string accompanied with SQL3189N has the index key column information.

6.2.5 Alter tables

As we have already stated, IXF files cannot preserve all attributes of the source Sybase tables. For example, referential constraints and foreign key definitions. Although our Sybase tables did not have any constraints and we did not have to perform this step in our project, you will probably need to execute `ALTER TABLE` statements to set some attributes that are not preserved by IXF files. The following example is adding a referential constraint:

```
alter table table1 add constraint const1 foreign key (colb)
references parenttable (cola) on delete cascade on update no action
```

Chapter 7. Application conversion

In this chapter, we discuss the application conversion from a Sybase environment to a DB2 UDB environment. We cover the following topics:

- SQL statements
- Transaction
- Built-in functions
- Declared temporary tables
- Save points
- Global variables
- Stored procedures
- Embedded SQL programs
- Client-Library programs

7.1 SQL statement comparison

This section compares the SQL statements of Sybase to that of DB2 UDB Version 7.1. Though many statements are alike, each has its own extensions and variations. We will look at only those statements that Sybase supports in their Transact-SQL (T-SQL). T-SQL supplied by Sybase with the Sybase Adaptive Server is an extension to the SQL language. Our primary focus will be on the insert, delete, update, and select statements.

7.1.1 Sybase naming conventions

First we will discuss the naming conventions for Sybase database tables. The format for the fully qualified table name is:

```
[database].[owner].table_name
```

Where:

- `database` is the name of the database. A query can join more than one table in different databases or in the same database.
- `owner` is the table owner. This will be the user-id of the person who created the table. You can also use the character string `dbo` to indicate database owner.
- `table_name` is the name of the table.

For DB2 UDB the table name can be qualified with the schema name as in:

```
schema.table_name
```

- `schema` is a grouping of logical database objects

- `table_name` is the name of the table

DB2 UDB can also join tables in other databases using the Relational Connect feature. Relational Connect is a very powerful feature which also allows native read access to other DBMS databases.

For the sake of clarity, we will use the phrase `table_name` from this point forward to indicate the fully qualified name of the table.

Access to the server and the database is different between Sybase and DB2. To access a server and database within Sybase, execute the following from an operating system command line:

```
isql -Uuser -Ppswd -Sserver
use database_name
go
select ....
```

This set of commands is executed from a UNIX command line. The `isql` command is the utility to interface with the Sybase server and the user, password and server information must be supplied. To connect to a database, you should issue the `use database_name` command. You can then begin to use the T-SQL statements.

To access an instance and/or database in DB2 UDB:

```
connect to database user user_id using pswd
select ....
```

This will connect you to the database named, and you must supply the user name. If `using` is omitted, you will be prompted for your password.

7.1.2 Insert statement

The basic format for the T-SQL insert statement is:

```
insert [into] table_name (column list)
   {values( expression1, expression2, ...)}
   |{ select statement}
```

Where:

- `into` for T-SQL is optional, required for DB2 UDB.
- `table_name` is the table or view name to insert into.
- `values` to be inserted can either be a list of columns and their corresponding values, or just a list of values if a value is supplied for all columns which do not allow null values.

- The select statement can be specified to insert some or all columns from another table.

The basic format for the DB2 insert statement is:

```
insert into table_name (column list)
   {values( expression1, expression2, ...)}
   |{ select statement}
```

Where:

- `into` is required for DB2 UDB.
- `table_name` is the table or view name to insert into.
- `values` to be inserted can either be a list of columns and their corresponding values, or just a list of values if a value is supplied for all columns which do not allow null values.

T-SQL and DB2 differences

- DB2 requires the word `into` and with T-SQL it is optional.
- Identity columns are handled differently:
  - DB2 will accept a value for an identity column if the column definition is *generated by default*. If the column is *generated always*, an error will be returned. Generated always is the recommendation when converting from Sybase.
  - T-SQL will not accept a value for an identity column except under one special circumstance. The table owner, database owner or the system administrator can execute `set identity_insert on` and insert a value. When `identity_insert` is `on`, all inserts to tables with identity columns must specify a value for the identity column(s).
- Keep in mind the default column definition differences between Sybase and DB2 UDB. Sybase does not allow null values by default; DB2 allows null values by default.
- T-SQL does not allow inserts on join views created `with check option` specified in the view, whereas DB2 will.
- T-SQL and DB2 UDB allow an optional `select from` clause to allow the user to insert values from other tables.
- For T-SQL when chained mode is set on and no transaction is active, the insert begins a transaction, and you must either commit or rollback the transaction.

7.1.3 Delete statement

The basic format for the T-SQL delete statement is:

```
delete[from] tablename
   {where search_condition }
```

Where:

- `from` for T-SQL is optional, required by DB2 UDB.
- `tablename` is the table or view name to delete from.
- `where` search_condition allows the user to delete rows which meet specific conditions.

The basic format for the DB2 delete statement is:

```
delete from tablename
   {where search_condition }
```

Where:

- `from` is required by DB2 UDB.
- `tablename` is the table or view name to delete from.
- `where` search_condition allows the user to delete rows which meet specific conditions.

Differences between T-SQL and DB2 UDB include:

- T-SQL does not allow deletes from a view that joins tables, even though the tables may individually allow deletes.
- T-SQL treats `from` as an option, DB2 requires that `from` be specified.
- T-SQL allows joined tables in the delete command as follows:

```
delete table02 from table02 t2, table03 t3 where t2.col1=t3.col1
```

This example allows data to be deleted from `table02` with matching data in `table03` on `col1`.

- The same result could be attained in DB2 UDB by coding this as a subquery:

```
delete from table02   where  col1 =  (select  t2.col1 from table02 t2,
          table03 t3 where t2.col1 = t3.col2);
```

- For T-SQL when chained mode is set on and no transaction is active, the delete begins a transaction and you must either commit or rollback the transaction.

7.1.4 Update statement

The basic format for the T-SQL update statement is:

```
update tablename
   set col1_name = expression1|NULL | select statement
   {where search_condition }
```

Where:

- `tablename` is the table or view name to update.
- `set` indicates the column(s) we want to change and the new value.
- `select` indicates that you can update a column or columns with values from other tables.
- `where search_condition` allows the user to update only rows which meet specific conditions.

The basic format for the DB2 update statement is:

```
update tablename
   set col1_name = expression1|NULL | DEFAULT | select statement
   {where search_condition }
```

Where:

- `tablename` is the table or view name to update.
- `set` indicates the column(s) we want to change and the new value.
- `select` indicates that you can update a column or columns with values from other tables.
- `where search_condition` allows the user to update only rows which meet specific conditions.

Differences between T-SQL and DB2 UDB include:

- In DB2 you can specify the default value to be used based on how the corresponding column is defined in the table. T-SQL does not allow you to specify this option.
- For T-SQL when chained mode is set on and no transaction is active, the update begins a transaction and you must either commit or rollback the transaction.

7.1.5 Select statement syntax

The format for the T-SQL select statement is:

```
select [ all | distinct | select _list
```

```
[into tablename]|
[from tablename
[holdlock | noholdlock] [shared],
[database].[owner].tablename2
where search_condition
group by ....
having .....
order by ....
compute ....
for {read only | update.....}
for browse
```

This is a list of the major functions and we will address each one separately:

- `all`, `distinct`, `select_list` works as in DB2 UDB. The `all` option is the default for both T-SQL and DB2 UDB.
- `into` is a T-SQL option to create a duplicate table or a subset based on the where clause. DB2 UDB supports the `into` option, but it works totally differently. For the DB2 UDB the `select into` produces a table consisting of at most one row with that row having the values of host variables and must be imbedded in an application program.

  The same results as the `into` option of the Sybase select statement can be attained with DB2 UDB as follows:

  ```
  create table02 like table01;
  insert into table02 select * from table01;
  ```

  This will create the table and then it can be populated using the insert. If you wish to create a subset you can add a where clause to the select statement to select only those rows you want.

  You can also create a table with only selected columns as follows:

  ```
  create    table table02 as
    (select  col1, col2  from table01)
     definition only;
  ```

  This will create a new table `table02` using the columns `col1` and `col2` from `table01`. The option `DEFINITION ONLY` would not insert the data.
- `tablename` is the table or view name to retrieve data.
- `where search_condition` search criteria to meet specific conditions.
- `[holdlock | noholdlock] [shared]` indicate the optional locking strategy to use. Shared can be used only in a select statement within a cursor. The holdlock makes a shared lock on the specified table or view and holds for the duration of the transaction. For DB2 UDB the same effect would be to set the isolation level to ‘RR’, Repeatable Read.

- `tablename2` indicates second or subsequent tables.
- `group by` works the same on both platforms.
- `having` is an option in both T-SQL and DB2 UDB that allows conditions to be set for the group by clause.
- `order by` is the same in both platforms.
- `compute` is a T-SQL only feature that allows creation of additional rows with summary values, when used with the row aggregate functions (`sum`, `avg`, `min`, `count`, `max`). DB2 UDB provides the `ROLLUP` which produces the same results but slightly different presentation.
- `read only`, `update` or `for browse` allow cursors to determine the type of locking and with DB2 UDB this will be done by using the read-only and for update clause on the select statement to accomplish the same results.

7.1.6 Select statement differences

In addition to the basic select statement, there are some other options that need to be discussed because of major differences in syntax.

7.1.6.1 Column headings

To change the name of a heading within T-SQL there are three methods:

1. select 'name' = col01
2. select col01 'name'
3. select col01 **as** 'name'

DB2 UDB supports only the third format.

7.1.6.2 Local variables

T-SQL allows you to declare local variables within a transaction and assign the variable a value. This would look like the following:

```
declare @department char(4)
select @department = 'x104'
select name, address from employee_db where dept = @department
```

The above SQL code can be in a stored procedure, embedded SQL or SQL entered within the Sybase server interactively. They can be used as counters or to control `while` loops or `if-else` code.

With DB2 UDB, this would need to be accomplished within a stored procedure.

7.1.6.3 Convert clause and casting

T-SQL offers a built-in function `convert` to allow conversion from one data type to another, including date formats. This would be accomplished by, for example:

```
select convert(char(8), getdate(),1)
```

This would return the current date in the format mm/dd/yy.

The convert can be used also to convert one data type to another, for example, to print you might want to convert smallint data type to char.

```
select convert(char(4), col1) from table01
```

The above would convert the contents of `col1` into a character format

For DB2 UDB you would need to use a casting function to convert the data.

```
values ( CAST (current timestamp as date) )
select CAST (col1 as char) from table01
```

The first examples will return the date in mm-dd-yyyy format. The second example casts the column `col1` into character format. DB2 also provides casting functions using the same name as the data type to which the data is converted. The following statements will return the same results as the examples shown above:

```
values ( date(current timestamp) )
select char(col1) from table01
```

7.1.6.4 Wild card characters

Both T-SQL and DB2 UDB support wild cards in the compare strings. The support is the same but T-SQL supports more characters (Table 16).

Table 16. Wild card character comparisons

T-SQL	DB2 UDB	Meaning
%	%	Used to represent a string of 0 or more characters
_	_	Used to represent a single character
^	n/a	Negative wildcard. Opposite of %
[ ]	n/a	Used to specify a range of characters. For example, [a-z] would be any character of a through z

The carot '^' and brackets '[ ]' are not supported by DB2 UDB, so these will have to be handled in other ways.

For the conversion, the carot could be changed to use `NOT LIKE '%abc%'` instead of `LIKE '^abc^'`.

Brackets could be handled with a substring function. Sybase example

```
select name from table01 where name like '[A-C]%'
```

This would return all rows where the first character is A, B, or C.

For DB2 UDB we could code this:

```
select name from table01 where substr(name, 1, 1) between 'A' and 'C'
```

For both T-SQL and DB2 UDB, the wild card characters are used in conjunction with the like clause.

7.1.6.5 Join

The join works basically the same in both T-SQL and DB2 UDB but have very different syntax. Beginning with SYBASE ASE release 12.0, there is support for the DB2 UDB format as well as the alternate format. Prior to release 12.0 the ***** indicated the outer join operator.

T-SQL supports joining multiple tables using such join operators as **=, >, <, <=** plus others. These work in the same manner as DB2 UDB.

For outer joins, T-SQL supports a different format with the ***** indicating the outer join operation.

Inner joins

The inner join works the same for both T-SQL and DB2 UDB with a syntax exception. T-SQL does an implicit join when two tables are named in the `select` statement. DB2 UDB also does the implicit inner join, but `INNER JOIN` can optionally be specified .

Outer joins

The T-SQL syntax for left outer joins is as follows:

```
select t1.col1, t1.col2, t5.col1 from table01 t1, table05 t5
   where t1.col1 *= t5.col1
```

The DB2 UDB syntax will require that you specify `LEFT OUTER JOIN` in the select statement to accomplish the same results.

```
select t1.col1, t1.col2 t1.col5 from table01
   left outer join table05 t5 on t1.col1 = t5.col1
```

Right outer joins would be:

```
select t1.col1, t1.col2, t5.col1 from table01 t1, table05 t5
   where t1.col1 =* t5.col1
```

Again, DB2 UDB would require `RIGHT OUTER JOIN` to obtain the same results.

```
select t1.col1, t1.col2 t1.col5 from table01
   right outer join table05 t5 on t1.col1 = t5.col1
```

7.1.6.6 Union

The UNION operator works the same on both platforms.

7.2 Transaction comparison

This section will be a brief comparison of the transaction model and the supported isolation level in Sybase and DB2 UDB.

7.2.1 Transaction model

In Sybase, a transaction is defined by enclosing SQL statements and/or system procedures within the phrases `BEGIN TRANSATION`, `SAVE TRANSACTION` (savepoint), `COMMIT`, and `ROLLBACK`. Sybase has two modes of transactions: *unchained mode* and *chained mode*. In T-SQL, you can change the transaction mode with the `SET CHAINED ON` or the `SET CHAINED OFF` command. The default mode is unchained mode in T-SQL. In unchained mode, you need to issue the explicit `BEGIN TRANSACTION` statement paired with the `COMMIT TRANSACTION` or `ROLLBACK TRANSACTION` statement to complete a transaction. If you do not describe the `BEGIN TRANSACTION` statement, each statement executed is implicitly committed. In chained mode, Sybase starts a transaction implicitly before the following statements: `DELETE`, `INSERT`, `OPEN`, `FETCH`, `SELECT`, and `UPDATE`. You need to issue the `COMMIT TRANSACTION` or `ROLLBACK` statement to close the transaction. The default mode in embedded SQL programs is chained mode.

In DB2 UDB, explicit transaction is not required, and implicit transaction is the only mode available.

Between Sybase and DB2 UDB, the transaction models are different. Sybase supports nested transactions, while DB2 UDB does not.

See the Sybase example shown in Figure 25. In this example, a program starts a transaction and then calls a stored procedure. The called stored procedure also starts a transaction. In this case, only the transaction started in the stored procedure will be committed if you issue a `COMMIT TRAN` command in the stored procedure. As we will discuss in 7.5, “Save point” on page 147, you can also set a save point and rollback to the save point. The `ROLLBACK` statement without the save point name rolls back all the transactions, including the one started in the caller program.

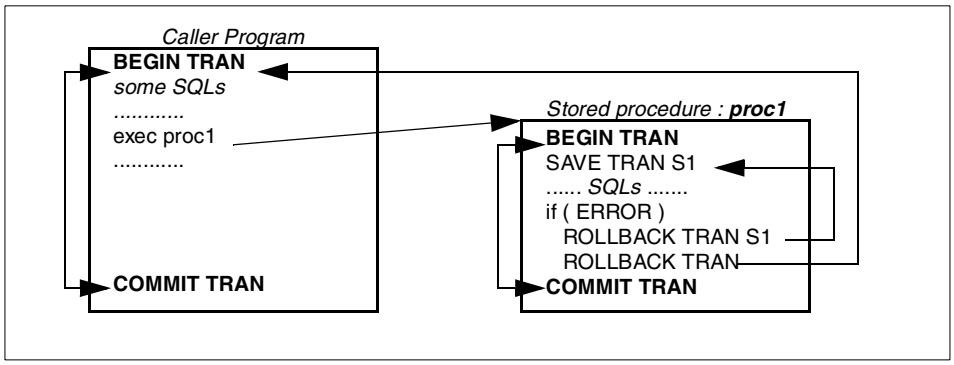

Figure 25. Nested transactions in Sybase

See the DB2 example shown in Figure 26. Since DB2 does not support nested transactions, if you issue the `COMMIT` or `ROLLBACK` statement in the called stored procedure like the Sybase example, all the changes done both in the caller program and the called stored procedure will be committed or rolled back.

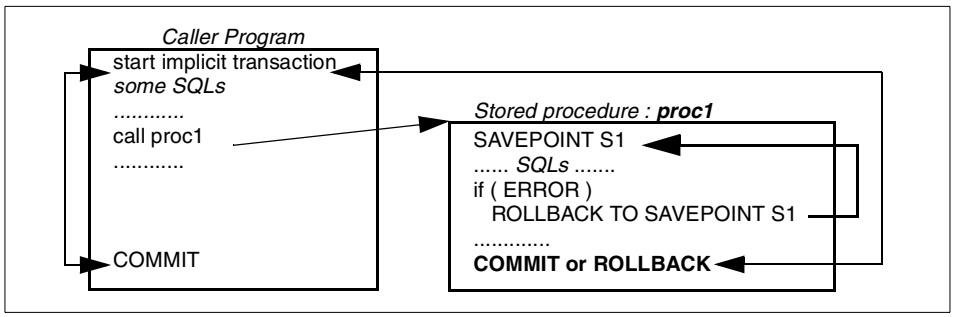

Figure 26. Transactions in DB2 UDB

7.2.2 Transaction isolation level

This section offers a brief comparison of the transaction isolation levels supported by Sybase and DB2. Table 17 lists the isolation levels supported by each platform.

Table 17. Transaction isolation levels

Sybase	DB2 UDB	Description
0	UR	Uncommitted Reads
1	CS	Cursor Stability, prevents uncommitted or 'dirty' reads
2	RS	Read Stability prevents rows from a transaction from being read by another transaction until committed
3	RR	Repeatable Read prevents *phantom* reads in addition to providing Read Stability

Sybase allows the isolation level to be changed at any time except within a transaction. The `holdlock` option on the select statement also causes the isolation level 2 to be used.

Setting the isolation level

Setting the isolation level in Sybase can be done by the `SET TRANSACTION ISOLATION LEVEL` command as follows:

```
set transaction isolation level 3
```

This example uses the set command to set the isolation level to 3

In DB2 UDB, the isolation level can be set only at specific times, based on the method with which it is set.

For embedded SQL: the isolation level is determined when the `PREP` or `BIND` command is executed:

```
PREP program1.sqc ISOLATION UR
BIND program1.bnd ISOLATION UR
```

These examples set the isolation level to Uncommitted Read which will allow uncommitted reads.

When using the Command Line Processor (CLP) the isolation level is set or changed by the `CHANGE ISOLATION` command:

```
CHANGE ISOLATION rr
```

This example sets the isolation level to Repeatable Read.

For DB2 Call Level Interface (DB2 CLI), you can use the `SQLSetConnectAttr` function with the `SQL_ATTR_TXN_ISOLATION` attribute at run time. This will set the transaction isolation level for the current connection referenced by the connection handle. The accepted values are:

SQL_TXN_READ_UNCOMMITTED - Uncommitted Read
SQL_TXN_READ_COMMITTED - Cursor Stability
SQL_TXN_REPEATABLE_READ - Read Stability
SQL_TXN_SERIALIZABLE - Repeatable Read

You can also use the `TXNISOLATION` keyword of the DB2 CLI configuration file (`db2cli.ini`) as follows:

```
TXNISOLATION=1
```

This would set the isolation level to Uncommitted Read (1). Other values allowed are 2=Cursor Stability, 4=Read Stability and 8=Repeatable Read.

7.3 Function comparison

This section discusses the functions available in Sybase and how they map to DB2 UDB functions. This section is divided into the following topics:

- Sybase functions that map to DB2 UDB
- Sybase functions that have no DB2 UDB equivalent
- Additional DB2 UDB Version 7.1 functions

For Sybase functions that have no DB2 UDB equivalent, we introduce some user defined functions with which you can implement the same functionality in DB2 UDB.

Here, we provide a general discussion of functions in Sybase and DB2 UDB. “Stored procedure conversion” on page 168 has examples with converted functions in our customer application.

7.3.1 Compatible functions

This section discusses compatible functions between Sybase and DB2 UDB.

7.3.1.1 Functions that have the same names

Table 18 is a listing of the Sybase functions that map to DB2 UDB functions. These functions are equivalent in name and functionality.

Table 18. Functions that map from Sybase to DB2 with same names

Sybase	DB2 UDB
ABS	ABS or ABSVAL
ACOS	ACOS
ASCII	ASCII
ASIN	ASIN
ATAN	ATAN
AVG	AVG
CEILING	CEILING or CEIL
COALESCE or ISNULL	COALESCE or VALUE
COS	COS
COT	COT
COUNT	COUNT
DEGREES	DEGREES
DIFFERENCE	DIFFERENCE
EXP	EXP
FLOOR	FLOOR
LOG	LOG
LOG10	LOG10
LOWER	LOWER or LCASE
LTRIM	LTRIM
MAX	MAX
MIN	MIN
NULLIF	NULLIF
POWER	POWER
RADIANS	RADIANS

Sybase	DB2 UDB
RAND	RAND
RIGHT	RIGHT
ROUND	ROUND
RTRIM	RTRIM
SIGN	SIGN
SIN	SIN
SOUNDEX	SOUNDEX
SPACE	SPACE
SQRT	SQRT
SUM	SUM
TAN	TAN
UPPER	UPPER or UCASE

7.3.1.2 Functions that have different names

Table 19 is also a listing of Sybase functions that map to DB2 UDB functions. These functions have different names, but are equivalent in functionality.

Table 19. Functions that map from Sybase to DB2 UDB with different names

Sybase	DB2 UDB
ATN2	ATAN2
CHAR	CHR
CHARINDEX(...,...)	LOCATE or POSSTR
CHAR_LENGTH	LENGTH
CONVERT(CHAR,...)	CHAR
CONVERT(CHAR(12),*datetime*,*style*)	CHAR(DATE(*timestamp*),*style*)[1]
CONVERT(DECIMAL(...,...),...)	DECIMAL or DEC
CONVERT(FLOAT(2),...)	DOUBLE_PRECISION, DOUBLE or FLOAT
CONVERT(IMAGE,...)	BLOB
CONVERT(INT,...)	INT or INTEGER

Sybase	DB2 UDB
CONVERT(SMALLINT)	SMALLINT
CONVERT(TEXT,..)	CLOB
CONVERT(VARCHAR,...)	VARCHAR
DATA_LENGTH	LENGTH[2]
DATEDIFF	TIMESTAMPDIFF[3]
DATENAME(MONTH,...)	MONTHNAME
DATENAME(WEEKDAY,...)	DAYNAME
DATEPART(DAY,...)	DAY
DATEPART(DAYOFYEAR,...)	DAYOFYEAR
DATEPART(HOUR,...)	HOUR
DATEPART(MILLISECOND,...)	MICROSECOND() * 1000
DATEPART(MINUTE,...)	MINUTE
DATEPART(MONTH,...)	MONTH
DATEPART(QUARTER,...)	QUARTER
DATEPART(SECOND,...)	SECOND
DATEPART(WEEKDAY,...)	DAYOFWEEK
DATEPART(WEEK,...)	WEEK
DATEPART(YEAR,...)	YEAR
INTTOHEX	HEX
LOG(...)	LN
REPLICATE	REPEAT
STUFF	INSERT[4]
STR	CHAR(DECIMAL(...))
SUBSTRING	SUBSTR
+	CONCAT or ||
%	MOD

1. In Sybase, you can use the CONVERT function to convert DATETIME or SMALLDATETIME data to a character type and specify the display format. Possible Sybase date formats are shown in Table 20.

Table 20. Possible Sybase date formats

Style number without century (yy)	Style number with century (yyyy)	Date format
N/A	0 or 100	mon dd yyyy hh:miAM (or PM)
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
6	106	dd mon yy
7	107	mon dd, yy
8	108	hh:mm:ss
N/A	9 or 109	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	mm-dd-yy
11	111	yy/mm/dd
12	112	yymmdd

In DB2, you can use the CHAR function to convert date type data to a character type and specify the display format. Possible DB2 date formats are shown in Table 21.

Table 21. Possible DB2 date formats

Format name	Abbreviation	Date format
International Standards Organization	ISO	yyyy-mm-dd
IBM USA standard	USA	mm/dd/yyyy
IBM European standard	EUR	dd.mm.yyyy

Format name	Abbreviation	Date format
Japanese Industrial Standard	JIS	yyyy-mm-dd
Installation-defined	LOCAL	Any installation defined form

The following example shows how to use the CONVERT function in Sybase:

```
1> SELECT CONVERT (CHAR(12),GETDATE(),101)
2> GO
------------
10/01/2000
```

And the following example shows how use the CHAR function in DB2:

```
db2 "VALUES ( CHAR ( DATE(CURRENT TIMESTAMP),USA )
1
----------
10/01/2000
```

2. The difference between DATA_LENGTH in Sybase and LENGTH in DB2 UDB

   In Sybase, DATA_LENGTH returns the number of bytes.

   In DB2 UDB, LENGTH also returns the number of bytes, except for a graphic string. The length of a graphic string is the number of DBCS characters.

3. Differences between DATEDIFF and TIMESTAMPDIFF include:

   These functions return the time difference between the two variables that mean date and time. Sybase has DATETIME data type that includes date, time and millisecond. DB2 UDB has TIMESTAMP data type that includes date, time and microsecond.

   The DATEDIFF function in Sybase and the TIMESTAMPDIFF function in DB2 UDB are similar, but the parameters and the output are different.

   The first parameter should be specified as shown in Table 22.

Table 22. First parameter for DATEDIFF and TIMESTAMPDIFF

Sybase	DB2 UDB
millisecond	1 *
second	2
minute	4
hour	8

Sybase	DB2 UDB
day dayofyear	16
week weekday colweekofyear coldayofweek	32
month	64
quarter	128
year calyearofweek	256

* The value 1 means microsecond, and you need to multiply the result of `TIMESTAMPFIFF` function by 1000.

The second parameter for DB2 UDB should be specified as shown here:

```
TIMESTAMPDIFF(16,CHAR(timestamp1 - timestamp2))
```

Also the output of `TIMESTAMPDIFF` is not exactly the same as in Sybase, because the following assumptions are used in estimating differences in timestamp in DB2 UDB:

- 365 days in a year
- 30 days in a month
- 24 hours in a day
- 60 minutes in an hour
- 60 seconds in a minute

For example, in DB2 UDB, if the number of days is requested for a difference in timestamps ‘1997-03-01-00.00.00' and ‘1997-02-01-00.00.00', the result is 30 as shown here.

In DB2 UDB:

```
db2 "VALUES(TIMESTAMPDIFF(16,CHAR(TIMESTAMP('1997-03-01-00.00.00')-
TIMESTAMP('1997-02-01-00.00.00'))))"
1
-----------
         30
```

Note that 16 is specified for the first parameter.

In Sybase:

```
1> SELECT DATEDIFF(DAY,"1997-02-01 01:00:00","1997-03-01 01:00:00")
2> GO

-----------
         28
```

Note: You can use a VALUES function to view values in DB2 UDB. It is equivalent to the SELECT statement without FROM clause in Sybase.

4. Difference in usage between STUFF and INSERT:

   See the examples shown in Table 23.

Table 23. Examples of conversion from STUFF to INSERT

	Sybase	DB2 UDB	RESULT
Case 1	STUFF('abc',2,3,'xyz')	INSERT('abc',2,3,'xyz')	axyz
Case 2	STUFF('abcdef',2,3,null)	INSERT('abcdef',2,3,'')	aef
Case 3	STUFF('abcdef',2,3,'')	INSERT('abcdef',2,3,' ')	a ef

Both the functions return a string where the argument3 bytes have been deleted from the argument1 beginning at the argument2 and where the argument4 has been inserted into the argument1 beginning at the argument2.

Both functions have the same functionality with small differences in the usage.

For the fourth argument, you cannot use the string 'NULL' to specify a null value in the INSERT function. You should use two single quotations without a space ('') to specify a null value as shown in the case 2 of Table 23. As shown in the case 3, two single quotations without a space ('') means a space character in the STUFF function. In INSERT function, you should use two single quotations with a space (' ') to specify a space character.

Creating user defined functions

If your application has many function calls whose names are different, but the functionality is the same between Sybase and DB2 UDB, changing all of them into the DB2 function name would be a considerable job. In this case, you may want to create source user defined functions in the DB2 database using the Sybase function name. By doing this, you can perform the application program conversion without changing the function calls one by one.

For example, if your Sybase application calls the ATN2 function, then the converted DB2 application should call the ATAN2 function, or you can leave

this function call if you create the user defined function ATN2 using the existing ATAN2 function. Calling the ATN2 function will invoke the ATAN2 function. Here is an example to create a user defined function ATN2:

```
CREATE FUNCTION ATN2 (DOUBLE,DOUBLE) RETURNS DOUBLE SOURCE SYSFUN.ATAN2
```

7.3.2 Sybase functions that have no DB2 UDB equivalent

There are functions in Sybase that are not in DB2 UDB.

7.3.2.1 Functions that can be implemented with SQL UDF

DB2 UDB V7.1 supports user-defined SQL functions (SQL UDF). They are much easier to create than legacy user-defined functions, since you do not need to program the functions using external programming language. Some of Sybase functions can be implemented using SQL UDF functions in DB2 UDB. Here we show examples for the following Sybase built-in functions:

- DATEADD
- COL_LENGTH
- PI

DATEADD function

The DATEADD function returns the DATETIME data type produced by adding a given number of years, quarters, hours, or other DATETIME parts to the specified DATETIME type data. Let us say you have a table tablea with a DATETIME type column cola. In Sybase, you can use the DATEADD function as follows:

```
SELECT DATEADD(DAY,5,cola) FROM tablea
```

This statement will add 5 days to the cola column and can be converted as following:

```
SELECT cola + 5 DAY FROM tablea
```

Generally SELECT DATEADD (DAY, *number*, *datetime column*) can be converted as SELECT *column* + *number* DAY, SELECT DATEADD (MONTH, *number*, *datetime column*) can be converted as SELECT *column* + *number* MONTH,and SELECT DATEADD (YEAR, *number*, *datetime column*) can be converted as SELECT *column* + *number* YEAR.

If you do not want to change all the SQL statements using the DATEADD function in your application, you can create the user defined function DATEADD featuring the same functionality as the Sybase built-in functin DATEADD and leave the SQL statements using the DATEADD function. The following example creates a user defined function DATEADD:

```
CREATE FUNCTION DATEADD(v_type VARCHAR(11),v_add INTEGER,v_date
TIMESTAMP)
    RETURNS TIMESTAMP
    LANGUAGE SQL
    NO EXTERNAL ACTION
    RETURN
    CASE UCASE(v_type)
--    'MILLISECOND' OR 'MS'
        WHEN 'MILLISECOND' THEN v_date + INT(v_add*1000) MICROSECOND
        WHEN 'MS'          THEN v_date + INT(v_add*1000) MICROSECOND
--    'SECOND' OR 'SS'
        WHEN 'SECOND'      THEN v_date + v_add SECOND
        WHEN 'SS'          THEN v_date + v_add SECOND
--    'MINUTE' OR 'MI'
        WHEN 'MINUTE'      THEN v_date + v_add MINUTE
        WHEN 'MI'          THEN v_date + v_add MINUTE
--    'hour' or 'hh'
        WHEN 'HOUR'        THEN v_date + v_add HOUR
        WHEN 'HH'          THEN v_date + v_add HOUR
--    'DAY' OR 'DD'
        WHEN 'DAY'         THEN v_date + v_add DAY
        WHEN 'DD'          THEN v_date + v_add DAY
--    'week' or 'wk'
        WHEN 'WEEK'        THEN v_date + INT(v_add*7) DAY
        WHEN 'WK'          THEN v_date + INT(v_add*7) DAY
--    'MONTH' OR 'MM'
        WHEN 'MONTH'       THEN v_date + v_add MONTH
        WHEN 'MM'          THEN v_date + v_add MONTH
--    'QUARTER'  OR 'QQ'
        WHEN 'QUARTER'     THEN v_date + INT(v_add*3) MONTH
        WHEN 'QQ'          THEN v_date + INT(v_add*3) MONTH
--    'YEAR'  OR 'YY'
        WHEN 'YEAR'        THEN v_date + v_add YEAR
        WHEN 'YY'          then v_date + v_add YEAR
    END
```

COL_LENGTH function

The following example creates a user defined function featuring the same functionality as the Sybase `COL_LENGTH` function:

```
CREATE FUNCTION COL_LENGTH (V_TABNAME VARCHAR(128),
   V_COLNAME VARCHAR(128))
        RETURNS INTEGER
        LANGUAGE SQL
        NO EXTERNAL ACTION
        RETURN
           SELECT LENGTH
```

```
FROM SYSCAT.COLUMNS
WHERE TABSCHEMA=USER
  AND TABNAME=V_TABNAME
  AND COLNAME=V_COLNAME
```

This function looks up the system catalog and obtains the length of the specified column name.

PI function

The `PI` function simply returns the constant value `3.1415926535897936`. The following example shows the conversion of the `PI` function to DB2 UDB:

```
CREATE FUNCTION PI ()
        RETURNS DOUBLE
        LANGUAGE SQL
        NO EXTERNAL ACTION
        DETERMINISTIC
        RETURN 3.1415926535897936
```

7.3.2.2 Functions that can be converted to special registers

The following functions can be converted to the special registers in DB2 UDB:

- USER
- GETDATE

Sybase's `USER` function returns the name of the current user. You can use the `USER` special register in DB2 UDB to implement the same functionality as the Sybase's `USER` function.

You can use the `CURRENT TIMESTAMP` special register in DB2 UDB for Sybase's `GETDATE` function.

USER function

The following examples show the use of the `USER` function in Sybase and the `USER` special register in DB2 UDB.

In Sybase:

```
1> SELECT USER
2> GO
------------------------------
 dbo
```

In DB2 UDB:

```
DB2 "VALUES(USER)"
1
------------------------------------------------------------------------
```

```
DB2INST1
```

GETDATE function

The following examples show the usage of the `GETDATE` function in Sybase and the `CURRENT TIMESTAMP` special register in DB2 UDB.

In Sybase:

```
1> SELECT GETDATE()
2> GO

 --------------------------
        Aug 17 2000  2:03PM
```

In DB2 UDB:

```
DB2 "VALUES(CURRENT TIMESTAMP)"
1
--------------------------
2000-08-17-14.09.16.751477
```

7.3.2.3 Functions that depend on the architecture of Sybase

These functions are depending on the architecture of Sybase. They are related to the object's id, physical architecture, role, license, and so on.

- COL_NAME
- CURUNRESERVEDPGS
- DATA_PGS
- DB_ID
- DB_NAME
- INDEX_COL
- INDEX_COLORDER
- IS_SEC_SERVICE_ON
- LCT_ADMIN
- LICENSE_ENABLED
- MUT_EXCL_ROLES
- OBJECT_ID
- OBJECT_NAME
- PROC_ROLE
- PTN_DATA_PGS
- RESERVED_PGS
- ROLE_CONTAIN
- ROLE_ID
- ROLE_NAME
- ROWCNT
- SHOW_ROLE

- SHOW_SEC_SERVICES
- SORTKEY
- SUSER_ID
- SUSER_NAME
- USED_PGS
- USER_ID
- USER_NAME
- VALID_USER

7.3.2.4 Functions converted to external functions

For these Sybase built in functions, You should write user defined functions (UDFs) with external programming language such as C.

- COMPARE
- HEXTOINT
- HOST_ID
- HOST_NAME
- PATINDEX
- REVERSE
- STR
- TEXTPTR
- TEXTVALID
- TSEQUAL
- VALID_NAME

Here are the steps that should be performed:

- Write the UDF
- Compile (and link) the UDF
- Register the UDF using the `CREATE FUNCTION` statement

See the *Application Development Guide*, SC09-2949 for the details.

7.3.3 Additional DB2 UDB Version 7.1 functions

There are a number of functions available in DB2 UDB V7.1 that are not in Sybase. These functions are summarized in Table 24.

Table 24. Functions Available in DB2 UDB V7.1

Function name	Description
BIGINT	Returns a 64 bit integer representation of a number or character string in the form of an integer constant.

Function name	Description
CORRELATION or CORR	Returns the coefficient of correlation of a set of number pairs.
COUNT_BIG	Returns the number of rows or values in a set of rows or values (column function). Result can be greater than the maximum value of integer.
COVARIANCE or COVAR	Returns the covariance of a set of number pairs.
DAYOFWEEK_ISO	Returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Monday.
DAYS	Returns an integer representation of a date.
DBCLOB	Casts from source type to DBCLOB, with optional length.
DEREF	Returns an instance of the target type of the reference type argument.
DIGITS	Returns the character string representation of a number.
GENERATE_UNIQUE	Returns a bit data character string that is unique compared to any other execution of the same function.
GRAPHIC	Cast from source type to GRAPHIC, with optional length.
GROUPING	Used with grouping-sets and super-groups to indicate sub-total rows generated by a grouping set (column function). The value returned is: 1 The value of the argument in the returned row is a null value and the row was generated for a grouping set. This generated row provides a sub-total for a grouping set. 0 otherwise.
HEX	Returns the hexadecimal representation of a value. This function includes the functionality of the `INTTOHEX` function that Sybase provides, and also returns character codes.
JULIAN_DAY	Returns an integer value representing the number of days from January 1, 4712 B.C. (the start of the Julian date calendar) to the date value specified in the argument.
LEFT	Returns a string consisting of the left most argument2 bytes in argument1.
LONG_VARCHAR	Returns a long string.
LONG_VARGRAPHIC	Casts from source type to LONG_VARGRAPHIC.

Function name	Description
MIDNIGHT_SECONDS	Returns an integer value in the range 0 to 86 400 representing the number of seconds between midnight and time value specified in the argument.
REAL	Returns the single-precision floating-point representation of a number.
REGR_AVGX	Returns quantities used to compute diagnostic statistics.
REGR_AVGY	Returns quantities used to compute diagnostic statistics.
REGR_COUNT	Returns the number of non-null number pairs used to fit the regression line.
REGR_INTERCEPT or REGR_ICPT	Returns the y-intercept of the regression line.
REGR_R2	Returns the coefficient of determination for the regression.
REGR_SLOPE	Returns the slope of the line.
REGR_SXX	Returns quantities used to compute diagnostic statistics.
REGR_SXY	Returns quantities used to compute diagnostic statistics.
REGR_SYY	Returns quantities used to compute diagnostic statistics.
REPLACE	Replaces all occurrences of argument2 in argument1 with argument3.
STDDEV	Returns the standard deviation of a set of numbers (column function).
TABLE_NAME	Returns an unqualified name of a table or view based on the object name given in argument1 and the optional schema name given in argument2. It is used to resolve aliases.
TABLE_SCHEMA	Returns the schema name portion of the two part table or view name given by the object name in argument1 and the optional schema name in argument2. It is used to resolve aliases.
TIME	Returns a time from a value.
TIMESTAMP	Returns a timestamp from a value or a pair of values.

Function name	Description
TIMESTAMP_ISO	Returns a timestamp value based on a date, time, or timestamp argument. If the argument is a date, it inserts zero for all the time elements. If the argument is a time, it inserts the value of CURRENT DATE for the date elements and zero for the fractional time element.
TRANSLATE	Returns a string in which one or more characters may have been translated into other characters.
TRUNC or TRUNCATE	Returns argument1 truncated to argument2 places right of the decimal point. If argument2 is negative, argument1 is truncated to the absolute value of argument2 places to the left of the decimal point. (argument2 is effectively the number of places to move).
TYPE_ID	Returns the internal data type identifier of the dynamic data type of the argument. Note that the result of this function is not portable across databases.
TYPE_NAME	Returns the unqualified name of the dynamic data type of the argument.
TYPE_SCHEMA	Returns the schema name of the dynamic type of the argument.
VARGRAPHIC	Returns a VARGRAPHIC representation of the first argument. If a second argument is present, it specifies the length of the result.
VARIANCE or VAR	Returns the variance of a set of numbers (column function).
WEEK_ISO	Returns the week of the year in of the argument as an integer value in the range of 1-53. The first day of a week is Monday. Week 1 is the first week of the year to contain a Thursday.

For the detailed information such as data types of the input parameters, see the *SQL Reference*, SC09-2974.

7.4 Declared temporary tables

Both Sybase and DB2 UDB create temporary tables which are used to place transient data for sorting rows, processing complex SQLs, and so on. They are automatically created and dropped by DBMS. User applications cannot manipulate them directly and these temporary tables do not persist beyond the current statement.

Applications can create their own temporary tables to store transient data as well. These tables are useful to use as intermediate tables to process large amount of data with complex queries. In this section, we discuss this type of temporary tables, which users can declare and use in application programs.

7.4.1 Temporary table comparison

Table 25 shows temporary space comparison between Sybase and DB2 UDB.

In Sybase these temporary tables are allocated in the `tempdb` database whereas in DB2 UDB, temporary tables for DBMS are allocated in system temporary table spaces, and user defined temporary tables are allocated in user temporary table spaces.

Sybase has two types of user defined temporary tables. One type is local temporary tables and the other type is global temporary tables. Local temporary tables (created in `tempdb`) are visible only from the current user who has created the temporary tables, and persist within the current application's connection. If local temporary tables are created in a stored procedure, they persist within a stored procedure's execution. The global temporary table has persistency between connections, and stays in `tempdb` until restarting of Sybase server. Sybase logs the changes on temporary tables in the `tempdb` database.

Table 25. Temporary space comparison between Sybase and DB2 UDB

	Sybase	**DB2 UDB**
Temporary space for DBMS is implemented in	tempdb database	System temporary table spaces
User defined temporary tables are created in	tempdb database	User temporary table spaces
User defined temporary tables	Local temporary tables	Declared temporary tables
	Global temporary tables	N/A

Temporary tables in DB2 UDB have the following characteristics:

- Do not exist as an entry in the system catalog tables
- Only used on behalf of a single application
- Changes are not logged
- Uses only minimal locking

In DB2, you can create a temporary table using the DECLARE GLOBAL TEMPORARY TABLE statement when you need it as an intermediate table. This type of temporary tables are called declared temporary tables. A declared temporary table persists within the current application's connection and is dropped automatically at the termination of the connection. During the connection period, the application can select rows from the declared temporary table, perform INSERT/UPDATE/DELETE statements without logging, and even drop the table explicitly. Declared temporary tables can be used from the standard SQL interfaces such as ODBC, CLI, and static/dynamic SQL.

You can use declared temporary table in DB2 UDB to implement Sybase's local temporary table. For a global temporary table in Sybase, you can use a regular table for the same purpose; however, you need to drop the table explicitly when you do not need it any longer.

7.4.2 Creating temporary tables

An example of the conversion from a local temporary table in Sybase to a declared temporary table in DB2 UDB is shown here:

Creating a local temporary table in Sybase
A table whose name start with "#" character is created as a local temporary table.

- With 1 SQL statement:

```
SELECT * INTO #temptab FROM tab1 WHERE col01>'C0000'
```

- With two SQL statements

```
CREATE #temptab(col01 CHAR(5), col02 CHAR(10))
INSERT * INTO #temptab SELECT * FROM TAB1 WHERE col01>'C0000'
```

Creating a declared temporary table in DB2 UDB
Use the DECLARE GLOBAL TEMPORARY TABLE statement to create a declared temporary table.

```
DECLARE GLOBAL TEMPORARY TABLE temptab(col01 CHAR(5),
   col02 CHAR(10)) NOT LOGGED;
INSERT INTO session.temptab SELECT * FROM tab1 WHERE col01>'C0000';
```

The NOT LOGGED option is mandatory for the DECLARE GLOBAL TERMPOARY TABLE statement.

To access a declared temporary table, you should use the schema name 'SESSION' with the table name.

You can use the existing table's column definition using the AS option or LIKE option of the DECLARE GLOBAL TEMPORARY TABLE statement as following:

```
DECLARE GLOBAL TEMPORARY TABLE temptab
  AS ( SELECT * FROM tab1) DEFINITION ONLY NOT LOGGED
```

or:

```
DECLARE GLOBAL TEMPORARY TABLE temptab LIKE tab1 NOT LOGGED
```

The first example creates a declared temporary table temptab using the SELECT statement. The second example creates the same declared temporary table temptab with a regular table tab1.

When you create a declared temporary table, the name should not be identical to the existing temporary table. However, you can replace the existing declared temporary table with the new one by specifying the WITH REPLACE option of the DECLARE GLOBAL TEMPORARY TABLE statement. For example, if you want to execute the same stored procedure more than once within a connection, and it creates a declared temporary table, you need to specify the WITH REPLACE option, or you need to drop the declared temporary table before creating the same declared temporary table. The declared temporary table will be dropped when the connection is terminated.

In Sybase, global temporary tables can be created. A table whose name starts with 'tempdb.' is created as a global temporary table. The following example creates a global temporary table:

```
select * into table tempdb..temptab from tab1 where col01>'C0000';
```

You can use a regular table in DB2 UDB for the purpose of a global temporary table in Sybase, although you need to drop the table explicitly when you do not need it any longer.

7.4.3 Considerations in declared temporary tables

Sybase logs the changes in temporary tables, whereas DB2 does not. Because of this difference, the ROLLBACK statement affects temporary tables differently on Sybase and DB2.

In Sybase, all the changes in temporary tables are logged, and issuing the ROLLBACK statement can back out the changes.

In DB2, any changes in temporary tables are not logged, therefore:

- If you modify the contents of a declared temporary table using the INSERT, UPDATE or DELETE statement within a transaction, and then roll back the

transaction by the ROLLBACK statement, or the statement fails, DB2 deletes all the rows of the declared temporary table.

- If you issue the ROLLBACK statement for the transaction that includes the DECLARE GLOBAL TEMPORARY TABLE statement, DB2 UDB drops the declared temporary table.
- If you issue the DROP TABLE statement for a declared temporary table, issuing the ROLLBACK statement for that transaction only restores an empty declared temporary table. Rolling back of the DROP TABLE statement does not restore the rows that existed in the declared temporary table.

There are also a couple of considerations when you use declared temporary tables.

Unless one or more cursors declared with the WITH HOLD option are still open on the declared temporary table, the default behavior of a declared temporary table is to delete all rows from the table when you commit a transaction. To avoid deleting all rows, you can use the ON COMMIT PRESERVE ROWS option of the DECLARE GLOBAL TEMPORARY TABLE statement.

In DB2 UDB, you need to define a user temporary table space for using declared temporary tables. Sybase creates both local temporary tables and global temporary tables in the tempdb database.

In Sybase, a global temporary table is treated almost the same way as a regular table. However, there are some considerations when using a local temporary table in Sybase. Table 26 shows the comparison in temporary tables between Sybase and DB2 UDB.

Table 26. Temporary tables comparison between Sybase and DB2 UDB

	Sybase	**DB2 UDB**
Can set referencial constraints?	No	No
Can define rules (Sybase) or check constraints (DB2 UDB)?	Yes	No
Can define the default value for a column?	Yes	Yes
Can create indexes on?	Yes	No
Can create views?	No	No
Can create triggers?	No	No
Can use user-defined type?	Yes•	No

- Only if user-defined data type is in the `tempdb..systypes`. If a user-defined data type has been explicitly created in the `tempdb` database since the last time the Server was started, you can use the user-defined data type with local and global temporary tables.

The differences shown in Table 26 are due to the fact that Sybase simply handles temporary tables like regular tables. For example, Sybase logs the changes on temporary tables. Temporary tables in DB2 are implemented to achieve high performance rather than the possibility to be recovered. Since temporary tables normally exist only temporarily, but high performance is required, we think this implementation is more suitable to this type of tables.

7.5 Save point

A save point is a mechanism of undoing work done by the DBMS when a database request fails.

If an error occurs during execution of statements, the save point can be used to undo changes made by the transaction between the time the save point was started and the time the save point rollback is requested.

Sybase supports multiple save points, and these can be named. In DB2 UDB Version 7.1, you can set only one save point at a time. If you want to set the second save point, you need to release the previous one. We still support the names for compatibility, with future DB2 UDB releases supporting multiple save points. The following examples show differences in the save point functionality between Sybase and DB2 UDB:

In Sybase:

```
BEGIN TRAN
   INSERT INTO TABLE01(COL01,COL02) VALUES('00001','AAAAA')
   SAVE TRAN A       -----   SAVEPOINT A
      INSERT INTO TABLE01(COL01,COL02) VALUES('00002','BBBBB')
   SAVE TRAN B       -----   SAVEPOINT B
      UPDATE TABLE01 SET COL02='CCCCC' WHERE COL01='00002'
```

At this point, you can execute `COMMIT`, `ROLLBACK`, `ROLLBACK TRAN A` (rollback to save point A) or `ROLLBACK TRAN B`. (rollback to save point B).

In DB2 UDB:

```
INSERT INTO TABLE01(COL01,COL02) VALUES('00001','AAAAA');
SAVEPOINT A ON ROLLBACK RETAIN CURSORS;
   INSERT INTO TABLE01(COL01,COL02) VALUES('00002','BBBBB');
RELEASE SAVEPOINT A;
```

```
SAVEPOINT B ON ROLLBACK RETAIN CURSORS;
   UPDATE TABLE01 SET COL02='CCCCC' WHERE COL01='00002'
```

At this point, you can execute COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT B, but you cannot execute ROLLBACK to SAVEPOINT A. If you do not execute the RELEASE SAVEPOINT A statement in this example, you will get an SQL error 'SQL20112N' when you set the save point B.

7.5.1 Setting a save point

In DB2 UDB, the following SQL statements enable you to create and control save points:

- SAVEPOINT - To set a save point, issue the SAVEPOINT statement. To improve the clarity of your code, you can choose a meaningful name for the save point. Options that can be specified in this statement include:
  - ON ROLLBACK RETAIN CURSORS - This option is mandatory, and indicates that the cursors are unchanged by a rollback to a save point unless data definition languages (DDL) are issued after the save point is set.
  - ON ROLLBACK RETAIN LOCKS - This option specifies system behavior upon rollback to the save point with respect to locks acquired after the setting of the save point. In DB2 UDB Version 7.1, this is the only option you can specify and the default. The acquired locks are not released when the ROLLBACK TO SAVEPOINT statement is executed. This is the same behavior as Sybase. The locks remain when the ROLLBACK TRAN *savepoint* statement is executed in Sybase.
- RELEASE SAVEPOINT - To release a save point, issue a RELEASE SAVEPOINT statement. If you do not explicitly release a save point with the RELEASE SAVEPOINT statement, it is released at the end of the transaction.
- ROLLBACK TO SAVEPOINT - To rollback to a save point, issue the ROLLBACK TO SAVEPOINT statement. The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the save point scope.
  - If DDLs or the SET INTEGRITY statements on which a cursor is dependent are executed after a save point is set, the cursor is made invalid when the ROLLBACK TO SAVEPOINT statement is executed.
  - Otherwise, if the cursor is referenced within the save point, the cursor remains open and positioned before the next logical row of the result table.
  - Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT.

  All locks are retained after the ROLLBACK TO SAVEPOINT statement.

ALL LOB locators are preserved following a ROLLBACK TO SAVEPOINT.

The following example shows the usage of save points:

```
INSERT INTO TABLE01 (COL01,COL02) VALUES (1,100);
INSERT INTO TABLE01 (COL01,COL02) VALUES (1,200);
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS;              (1)
INSERT INTO TABLE01 (COL01,COL02) VALUES (2,300);
INSERT INTO TABLE01 (COL01,COL02) VALUES (2,400);
ROLLBACK TO SAVEPOINT S1;                             (2)
SAVEPOINT S2 ON ROLLBACK RETAIN CURSORS;              (3)
INSERT INTO TABLE01 (COL01,COL02) VALUES (3,500);
INSERT INTO TABLE01 (COL01,COL02) VALUES (3,600);
COMMIT;
```

Contents of TABLE01:

```
COL01    COL02
-------  -------
      1      100
      1      200
      3      500
      3      600
```

This example includes two save points.

1. The first save point is named S1.
2. After setting the save point S1, the ROLLBACK TO SAVEPOINT statement is issued and two of the INSERT statements are roll backed within the save point scope between (1) and (2).
3. The new savepoint(3) which is named S2 is declared.

7.5.2 Considerations in using save points

DB2 UDB V7.1 places the following restrictions on your use of save points in applications:

- You cannot use save points within atomic compound SQL.
- You cannot use atomic compound SQL while a save point is set. Sybase does not have the function equivalent to compound SQL in DB2 UDB.
- You cannot use a save point while another save point is set. Sybase supports nested save points.
- You cannot use save points in triggers while Sybase supports save points in triggers.

7.6 Sybase's global variable

Sybase has many global variables, which are system-defined variables that Sybase server updates on an ongoing basis. You can query global variables to monitor system activities or get connection information. For some global variables, you can also set a value using the `SET` command to control query processing of the current session.

To monitor system activities, you can use the Snapshot Monitor in DB2 UDB. DB2 UDB also has special registers, which are also updated on an ongoing basis by DB2 UDB. For some special registers, you can use the `SET` command to update the value.

Some Sybase global variables can be mapped to the data element which you can obtain using the Snapshot Monitor in DB2, others can be mapped to DB2 special registers or different features of DB2 UDB depending on the purpose of the global variable. In this section, we show you how you can implement the functionality of the following global variables in DB2 UDB:

- @@connections
- @@error
- @@sqlstatus
- @@identity
- @@parallel_degree
- @@rowcount

7.6.1 The @@connections global variable

The `@@connections` global variable returns the number of logins or attempted logins to the Sybase server since it was started.

In DB2, you can execute the Snapshot Monitor and get the 'Connects Since Database Activation' element, which is indicated as 'Application connects' in the output, as the following example:

```
$ db2 get snapshot for database on dbname | grep 'Application connects'
Application connects                         = 24
```

The Snapshot Monitor returns the number of connected user since the database was activated. DB2 activates a database when the first application

is connected to the database or you execute the ACTIVATE DATABASE command for the database.

Be aware that the value obtained from the Snapshot Monitor does not include the number of the failed connection, and also the value is counted at database level since each connection is established to a database in DB2.

7.6.2 The @@error and the @@sqlstatus global variables

In Sybase, you can query the @@error or @@sqlstatus global variables to check whether the previous statement succeeded or not. The @@error has the error status of the most recently executed statement. The @@sqlstatus has the status from the last FETCH statement. Its value is 0 (success), 1 (error), or 2 (no more data).

In DB2 UDB, return codes from SQL statements are handled with SQLCODE or SQLSTATE. You can use either of them in applications. The SQLCODE conforms to ISO/ANSI SQL standard, and the SQLSTATE is based on the ISO/ANSI SQL92 standard.

Details about error handling in SQL Stored Procedures is described in 7.8.2.1, “Error handling” on page 170.

7.6.3 The @@identity global variable

In Sybase, the @@identity global variable contains the last identity value inserted into an identity column in the current user session.

You can use the IDENTITY_VAL_LOCAL function in DB2 UDB for the same purpose.

The value returned from the IDENTITY_VAL_LOCAL function is the value assigned to the identity column of the table identified in the most recent single row INSERT statement with a VALUES clause for a table containing an identity column.

To update the value returned to the IDENTITY_VAL_LOCAL function, the INSERT statement must be a single row INSERT statement with a VALUES clause that is issued for a table containing an identity column as the following example:

```
INSERT INTO table01 (col01,col02) VALUE (default,1)
```

In this example, it is assumed that the column col01 is the identity column of the table table01.

The following statements will not affect the value returned to the IDENTITY_VAL_LOCAL function because they are not single row insert statements with a VALUES clause:

```
INSERT INTO table01 (col01,col02) VALUE (default,1),(default,2)
INSERT INTO table01 (col02) SELECT col03 from table02
```

In Sybase, the INSERT statement does not need to be a single row INSERT statement with a VALUES clause for the @@identity variable to get the inserted identity value. The INSERT statement with the SELECT statement, as in this example, will update the value of the @@identity variable:

```
INSERT INTO TABLE01(COL01) SELECT COL01 FROM TABLE02
```

Note that the IDENTITY_VAL_LOCAL function gets the identity value which was assigned by the most recent INSERT statement within the same program or stored procedure. For example, if an identity value is generated by the INSERT statement in a stored procedure, the IDENTITY_VAL_LOCAL function called in the caller program cannot get the inserted identity value. You need to use the identity_val_local function in the same stored procedure to get the inserted identity value. Sybase has the same consideration.

The IDENTITY_VAL_LOCAL function in DB2 UDB returns the null value when a COMMIT or ROLLBACK of a unit of work has occurred since the most recent INSERT statement that assigned a value.

In Sybase, the updated @@identity global variable is preserved even if the ROLLBACK statement is issued. If you insert 10 rows into the table that has an identity column and execute the ROLLBACK statement, the @@identity variable will have the value 10.

The following example shows the usage of the @@identity global variable in Sybase.

Create table:

```
CREATE TABLE TABLE01
   (IDCOL NUMERIC(5,0) IDENTITY,
    COL01 CHAR(5))
```

Sample operation:

```
INSERT INTO TABLE01 (COL01) VALUES("00001")
(1 row affected)
SELECT @@IDENTITY
----------------------------------------
                                       1
INSERT INTO TABLE01 SELECT COL01 FROM TABLE02
```

```
(4 rows affected)
SELECT @@IDENTITY
----------------------------------------
                                       5
```

The following example shows the use of IDENTITY_VAL_LOCAL function in DB2 UDB.

Create table:

```
CREATE TABLE TABLE01
  (IDCOL INT NOT NULL GENERATED ALWAYS AS IDENTITY
    (START WITH 1,INCREMENT BY 1),
   COL1 CHAR(5));
```

Sample operation:

```
db2 +c "INSERT INTO TABLE01(COL1) VALUES('00001')"
db2 +c "VALUES(IDENTITY_VAL_LOCAL())"
--------------------------------
                              1.
db2 +c "INSERT INTO TABLE01(COL01) SELECT COL01 FROM TABLE02"
db2 +c "VALUES(IDENTITY_VAL_LOCAL())"
--------------------------------
                              1.
```

Note that you must execute these operations with the auto-commit mode set off. Otherwise the INSERT statement is automatically committed and the IDENTITY_VAL_LOCAL function returns the null value. In this sample, "+c" option is used to set the auto-commit mode off.

The second INSERT statement does not affect the result of IDENTITY_VAL_LOCAL function, because this statement is not a single row INSERT statement with a VALUES clause.

7.6.4 The @@parallel_degree global variable

Both Sybase and DB2 UDB have a capability to exploit multiple processors to execute complex SQL requests effectively.

In Sybase, the @@parallel_degree variable contains the current maximum parallel degree setting.

You can use CURRENT DEGREE special register in DB2 UDB for the same purpose. Both the value of @@parallel_degree variable and the CURRENT DEGREE special register can be set and retrieved. The following examples show the

usage of the @@parallel_degree global variable and the CURRENT DEGREE special register:

In Sybase:

```
1> set parallel_degree 1
2> go
1> select @@parallel_degree
2> go
-----------
          1
```

In DB2 UDB:

```
db2 "set current degree = '2'"
db2 "values(current degree)"
1
-----
2
```

In DB2 UDB, the CURRENT DEGREE special register specifies the degree of intra-partition parallelism for the execution of dynamic SQL statements. Valid values are 'ANY' or the string representation of an integer between 1 and 32767. For static SQL statements, you can use the DEGREE option of the BIND command to set the query degree.

If the value of CURRENT DEGREE is 'ANY', the execution of dynamic SQL statements will involve intra-partition parallelism using a degree determined by the database manager.

7.6.5 The @@rowcount global variable

In Sybase, the @@rowcount variable contains the number of rows affected by the last query. The value represents the number of rows of a cursor result set returned to the client, up to the last fetch request.

In DB2 UDB, you can use the GET DIAGNOSTICS statement with the ROW_COUNT option in stored procedures. If the previous statement is the PREPARE statement, ROW_COUNT identifies the estimated number of result rows in the prepared statement. If the previous statement is the DELETE, INSERT, or UPDATE statement, ROW_COUNT identifies the number of rows deleted, inserted, or updated by that statement, excluding rows affected by either triggers or referential integrity constraints.

After the SELECT or the FETCH statement is executed, Sybase sets the number of affected rows to the @@rowcount variable, whereas the ROW_COUNT is not affected by the SELECT or the FETCH statement in DB2. Only after the PREPARE

statement, you can obtain the estimated number of rows using the GET DIAGNOSTICS statement. The GET DIAGNOSTICS statement derives the number of rows from the statistics information in the system catalog.

The following stored procedure uses the GET DIAGNOSTICS statement to get the number of affected rows:

```
CREATE PROCEDURE sqlprocg ( IN deptnbr VARCHAR(3),OUT rcount INTEGER)
   LANGUAGE SQL
   BEGIN
      DECLARE SQLSTATE CHAR(5)
      UPDATE CORPDATA.PROJECT
         SET PRSTAFF = PRSTAFF + 1.5
         WHERE DEPTNO = deptnbr;
      GET DIAGNOSTICS rcount = ROW_COUNT;
END
```

This stored procedure issues the UPDATE statement and the GET DIAGNOSTICS statement, and returns the affected row count.

In Sybase you can use the @@rowcount variable to set the maximum number of rows that will be retrieved in the following SQL queries, with 0 meaning all of them.

DB2 UDB does not have such a variable; however, you can set the maximum number of retrieved rows using "FETCH FIRST n ROWS ONLY" option of the SELECT statement.

The following examples show the usage of the @@rowcount global variable and the FETCH FIRST n ROWS ONLY option of the SELECT statement:

In Sybase:

```
set rowcount 100
select * from table01
```

In DB2 UDB:

```
select * from table01 fetch first 100 rows only
```

These two examples above returns only the first 100 rows from the table table01.

7.7 Trigger conversion

A trigger defines a set of actions that are activated, or triggered, by an insert, update, or delete on specified base table. Triggers are powerful tools that can

be used for several purposes. They can be implemented to support general forms of integrity such as business rules; they can be used to make sure that whenever a certain action occurs in a database, another action – or actions – automatically follows. In general, triggers can be used to enforce the validity of data and to capture rules that involve transitional business rules, in other words, rules that involve different states of the data. In this section, we discuss the trigger conversion from Sybase to DB2.

7.7.1 Sybase and DB2 triggers

Table 27 shows differences between Sybase triggers and DB2 triggers.

Table 27. Sybase - DB2 trigger differences

	Sybase	**DB2**
Maximum length for trigger name	30 characters	18 characters
Triggering Event	Insert, update or delete of rows in a specified table	Insert, update or delete of rows in a specified table
Activation time	AFTER insert, update or delete	BEFORE or AFTER insert, update or delete
Allowed SQL statements	All statements except DML type statements	Before triggers: The SET statements that modify the new row, `SELECT, VALUES, SIGNAL`. After triggers: `INSERT, DELETE, UPDATE, SELECT, VALUES, SIGNAL`
Granularity (activated once an SQL statement, or once for each row modified)	Statement level	Before triggers: row level (`FOR EACH ROW` option) After triggers: Statement level (`FOR EACH STATEMENT` option) or row level (`FOR EACH ROW` option)
Transition variables	`DELETED, INSERTED`	`NEW, OLD, NEW_TABLE, OLD_TABLE`
Nesting	16 levels	16 levels

As you can see in the table above, Sybase triggers always perform the trigger action after the changes caused by the actual update of the subject table. DB2 trigger can perform the trigger action after or before the changes.

Sybase triggers can be activated only once an SQL statement, whereas DB2 triggers can be activated once an SQL statement or once for each row modified. In Sybase, a trigger needs to have a cursor using the logical tables `INSERTED` or `DELETED` to access individual rows.

Sybase allows triggers to have more various SQL statements than DB2 does, thus converting Sybase triggers into DB2 triggers is not always a trivial task.

In the following sections, we show four Sybase triggers and how we could convert them into DB2 triggers.

7.7.2 Conversion of an insert, update trigger

The first trigger example here fires on an insert or update to the `table01` table. In DB2, only one triggering event (insert, update, delete) may be specified per trigger. In order to convert this procedure two triggers will need to be created - one for the insert event and one for the update event.

Here is the Sybase trigger:

```
create trigger trig01
on table01 for insert, update
as
begin
  declare @savecount int
  select @savecount = @@rowcount
  --Trigger is not fired if the table is maintained by the maint user
  if suser_name() like "%maint"
                return
  if @savecount = 0
  return

  if not exists (select * from table02,inserted where
    col01 = "AA" and col02 = "BB")
  begin
    raiserror 88888 "BB does not exist."
    rollback transaction
    return
  end

  if not exists (select * from table02,inserted where
    col01 = "AA" and col02 = "CC")
  begin
```

```
    raiserror 99999 "CC does not exist."
    rollback transaction
    return
  end

end
```

Here is the DB2 UDB translation. We need to create two triggers, for insert and for update.

For insert:

```
CREATE TRIGGER trig01ins
AFTER  INSERT                                                    (1)
ON table01
REFERENCING NEW_TABLE as new                                     (2)
FOR EACH STATEMENT                                               (3)
MODE DB2SQL                                                      (4)
WHEN
((select USER from sysibm.sysdummy1 where USER like '%maint') is null
                                                                 (5)
and
( select count(*) from new) > 0))                                (6)
BEGIN ATOMIC                                                     (7)
VALUES (
  CASE WHEN (select count(*) from table02, new where
         col01 = 'AA' and col02 = 'BB') = 0
  THEN raise_error ('88888', 'BB does not exist.')               (8)
  WHEN (select count(*) from table02,new where
      col01 = 'AA' and col02 = 'CC') = 0
  THEN raise_error ('99999', 'CC does not exist.')
  else 0
  end);

end
```

Here is the trigger for update:

```
CREATE TRIGGER trig01upd
AFTER  UPDATE
ON table01
REFERENCING NEW_TABLE as new
FOR EACH STATEMENT
MODE DB2SQL
WHEN
((select USER from sysibm.sysdummy1 where USER like '%maint') is null
and
( select count(*) from new) > 0))
```

```
BEGIN ATOMIC
VALUES (
  CASE WHEN (select count(*) from table02, new where
         col01 = 'AA' and col02 = 'BB') = 0
  THEN raise_error ('88888', 'BB does not exist.')
  WHEN (select count(*) from table02,new where
      col01 = 'AA' and col02 = 'CC') = 0
  THEN raise_error ('99999', 'CC does not exist.')
  else 0
  end);

end
```

1. Since all Sybase triggers are — by definition — AFTER triggers, the triggered action `AFTER` will be consistent throughout the conversion of these triggers. Also, DB2 allows only one triggering event (`INSERT`, `UPDATE` or `DELETE`) to be specified for the triggering action.
2. Sybase uses two special temporary tables in trigger statements: the `deleted` table and the `inserted` table - these are used to temporarily preserve the effects of an insert, update or delete. In DB2, the row transition variables `NEW` and `OLD`, and the table transition variables `OLD_TABLE` and `NEW_TABLE` are used to reference the data values of rows or tables before or after a triggering event.
3. All Sybase triggers are statement level triggers. In most cases, converted triggers will remain as statement level triggers.
4. MODE DB2SQL: This is required. This clause is used to specify the mode of triggers. This is the only valid mode currently supported.
5. The `WHEN` clause evaluates to `True` or `False`. The associated action is performed only if the specified search condition evaluates as true. In this case the trigger tests to see if the user is `%maint`, or if no rows were affected by the triggering event. If these conditions evaluate to true, no triggering action is performed.
6. The transition table variable `NEW` is used to examine the number of rows modified by the insert statement; if there are no rows, no triggering action is performed.
7. BEGIN ATOMIC - END : If the trigger body consists of more than one SQL statement, the statements are enclosed between `BEGIN ATOMIC` and `END`. In this case it is optional, since there is only one SQL statement.
8. The `raise_error` function aborts the processing of the current SQL statement and raises an error condition. It also rolls back all database changes caused by the current SQL statement. The second parameter, a message string, is returned to the application program in the `sqlerrmc` field

of the SQLCA structure. This replaces rollback transaction in the Sybase trigger.

7.7.3 Conversion of cursor processing in Sybase triggers

Since Sybase triggers are all statement-level triggers, your Sybase source trigger might have a cursor to examine each row of the affected table. DB2 triggers can be row-level triggers, and you can use the row transition variable NEW to examine each row of the affected table. See the following Sybase trigger and how we could convert it into a DB2 trigger:

Sybase Source:

```
create trigger trg02
on table02 for insert, update
as
begin
   declare  @savecount int,
            @var01 varchar(10)
   select @savecount = @@rowcount

  --Trigger is not fired if the table is maintained by the maint user
  if suser_name() like "%maint"
               return
  if @savecount = 0
  return
  declare csr_inserted cursor for select col01 from inserted
  open csr_inserted
  fetch csr_inserted into @var01
  while @@sqlstatus = 0
  BEGIN
   /* Verify against transaction detail */
   IF @var01 != 'DEFAULT'
   BEGIN
     if not exists (select * from table03 a, inserted i
      where a.colaa = @var01
       begin
         raiserror 77777 "There is no record like this."
         rollback transaction
         return
       end
   END
   fetch csr_inserted into @var01
  END
end
```

Here is the DB2 conversion:

For insert:

```
CREATE TRIGGER trg02ins
AFTER  INSERT  ON table02
REFERENCING  NEW AS newrow  NEW_TABLE AS newtable
FOR EACH ROW  MODE DB2SQL                                              (1)
WHEN ( (select count(*) from newtable) > 0
        and (select USER from sysibm.sysdummy1
               where USER like '%maint') is null
        and (select newrow.col01 from newtable) != 'DEFAULT')
  BEGIN ATOMIC
  VALUES(
  CASE WHEN (select count(*) from table03 a, newtable n
        where a.colaa = newrow.col01) = 0
        THEN raise_error ('77777', 'There is no record like this.')
         else 0
        end);
  END
```

For update:

```
CREATE TRIGGER trg02upd
AFTER  UPDATE ON table02
REFERENCING  NEW AS newrow  NEW_TABLE AS newtable
FOR EACH ROW  MODE DB2SQL                                              (1)
WHEN ( (select count(*) from newtable) > 0
        and (select USER from sysibm.sysdummy1
               where USER like '%maint') is null
        and (select newrow.col01 from newtable) != 'DEFAULT')
  BEGIN ATOMIC
  VALUES(
  CASE WHEN (select count(*) from table03 a, newtable n
        where a.colaa = newrow.col01) = 0
        THEN raise_error ('77777', 'There is no record like this.')
         else 0
        end);
  END
```

- FOR EACH ROW: this specifies that the triggered action be applied once for each row of the subject table that is affected by the triggering SQL operation. Since Sybase triggers are all statement-level triggers the Sybase trigger needed to use a cursor to examine the value of the affected table in each row.

7.7.4 Conversion of Sybase delete triggers

The next example deletes rows from joined tables. DB2 does not allow joined tables in a delete statement; thus, the delete statement need to be replaced by a sub-query. See the following source Sybase trigger and how we converted it.

Sybase source:

```
create trigger trg03
on table04 for delete
as
begin

  --Trigger is not fired if the table is maintained by the maint user
  if suser_name() like "%maint"
               return

   delete table05 from table05 a,deleted b
   where a.col01 = b.col01
end
```

Here is the DB2 trigger converted from the Sybase source trigger:

```
CREATE TRIGGER trg03
AFTER  DELETE  ON table04
 REFERENCING  OLD_TABLE AS oldtable
 FOR EACH STATEMENT  MODE DB2SQL
WHEN ((select  USER from sysibm.sysdummy1 where USER like '%maint') is
null )
BEGIN ATOMIC
   delete from table05
   where  col01= (select col01 from oldtable);                    (1)
END
```

1. DB2 does not allow joined tables in deletes; this needed to be replaced by a subquery.

7.7.5 Conversion of Sybase triggers: if update (column name)

The next example is an update trigger that is fired when either the column `col01` or `col02` of the `table06` is updated. See the following Sybase source trigger and the DB2 trigger we converted.

Sybase Source:

```
create trigger trg04 on table06 for update as
begin
    declare @count int

    select @count = count(*) from inserted

    if @count = 0
   return

    if update(col01) or update(col02)
    begin
      rollback transaction
      raiserror 666666 'can not update col01 or col02'
      return
    end

    insert table07 (col01,col02)
       select getdate(), 'tried to update col01 or col02' from inserted
end
```

Here is the DB2 conversion:

```
CREATE TRIGGER trg04
AFTER UPDATE OF col01, col02 ON table06                          (1)
REFERENCING NEW_TABLE AS new
FOR EACH STATEMENT  MODE DB2SQL
WHEN ((select count(*) from new) > 0)

BEGIN ATOMIC
  SIGNAL SQLSTATE '66666' ('can not update col01 or col02');     (2)
  insert into table07 (col01,col02)
(select CURRENT TIMESTAMP,'tried to update col01 or col02' from new);

END
```

1. AFTER UPDATE OF <column Name>: The trigger will be activated by the update of a column identified in the column-name list.
2. SIGNAL SQLSTATE: the SIGNAL statement raises an error condition and rolls back the effects of an SQL statement; however, it leaves the transaction in progress so the user or application program can still choose to commit or rollback the other statements in the transaction. The message specified by the SIGNAL statement is returned to the application program in the sqlerrmc field of the SQLCA structure.

7.7.6 Creating triggers from the command line processor

To create a trigger from the DB2 command line processor (CLP), which you can start by typing `db2` from the operating system command prompt, you should first place the source code for the trigger into a file. The CLP uses the semicolon (;) as the default delimiter for statements, however if the triggers have compound statements, the statements within the compound statement are also terminated with semicolons, causing the CLP to interpret those semicolons as the end of the `CREATE TRIGGER` statement. This results in a syntax error.

To avoid this, use an alternative terminating character in your file, and change the CLP invocation command to identify this new character as the terminating character.

The following is an example to crate a trigger from the CLP. The file name is `myfile.trg` and the terminating character is an exclamation point (!):

```
CONNECT TO <database name> USER <username> USING <password>!
CREATE TRIGGER trg03
 AFTER  DELETE  ON table04
 REFERENCING  OLD_TABLE AS oldtable
 FOR EACH STATEMENT  MODE DB2SQL
 WHEN ((select  USER from sysibm.sysdummy1 where USER like '%maint') is
null )
 BEGIN ATOMIC
  delete from table05 where col01= (select col01 from oldtable);

END!    -- (!) is the terminating character

CONNECT RESET!
```

To execute the above file, you must invoke the CLP with the `-td` parameter to specify an exclamation point (!) as the terminating character, as follows:

```
db2 -td! -fmyfile.trg
```

7.7.7 Creating triggers from the Control Center

Triggers can also be created from the Control Center GUI. Perform the following steps:

1. Open the Control Center, expand the SYSTEMS, INSTANCES and DATABASES until you see the Database in which you want to create the trigger.
2. Expand the Database until you see Tables, Views, Aliases, Triggers. Select **Triggers**, right-click, then select **Create** as shown in Figure 27.

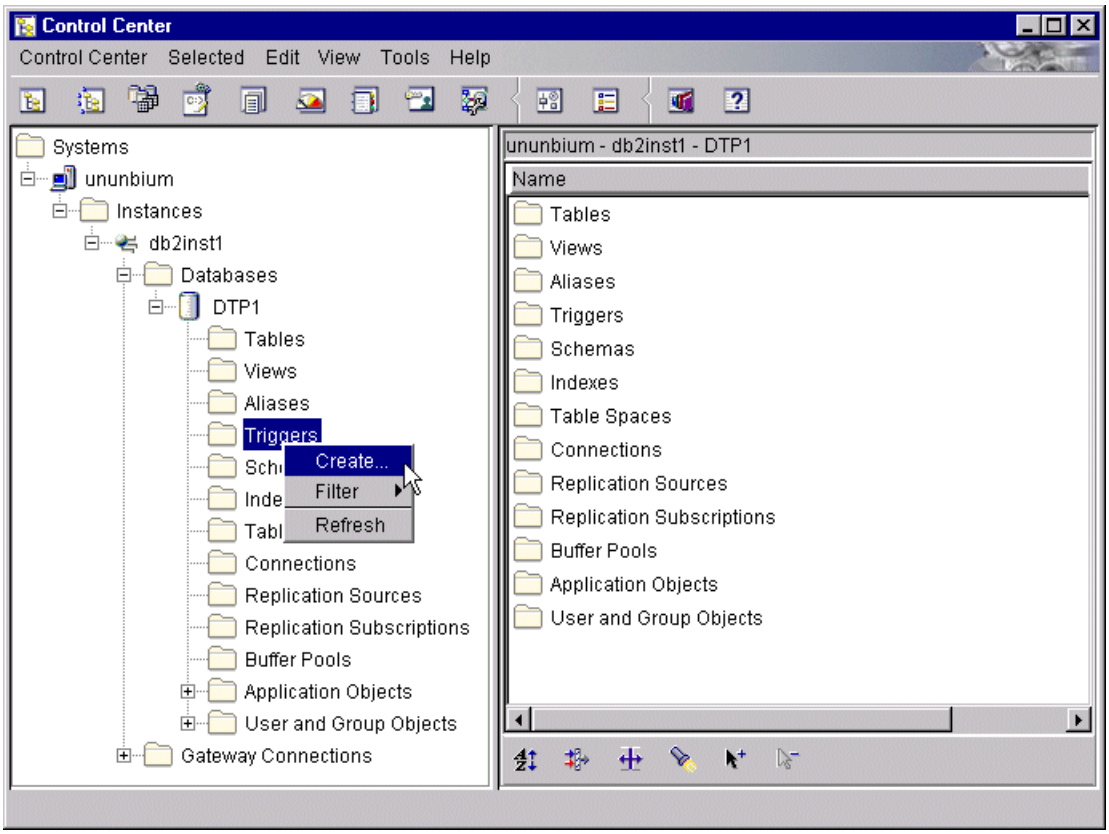

Figure 27. Creating a trigger from the Control Center (1)

3. You will be presented with the dialog as shown in Figure 28. Type the trigger name, select the table on which you want to create the trigger, and select the operation that fires the trigger.

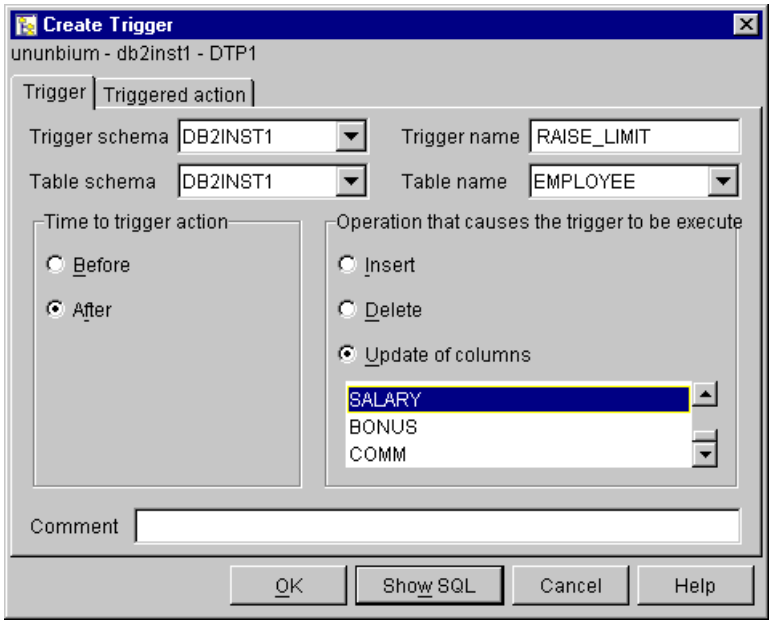

Figure 28. Creating a trigger from the Control Center (2)

4. Select the **Triggered Action** tab, and you will be presented with the dialog as shown in Figure 29. Depending on whether you are creating a BEFORE of AFTER Trigger, complete the correlation name, and/or Temporary table names sections; and enter the logic for the body of the trigger.

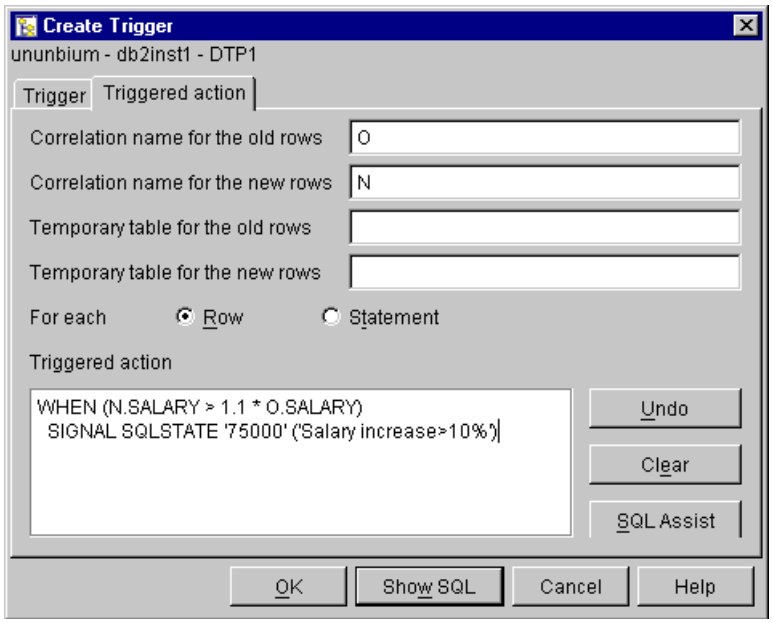

Figure 29. Creating a trigger from the Control Center (3)

5. When you are finished you may view the completed script by choosing **Show SQL** as shown in Figure 30.

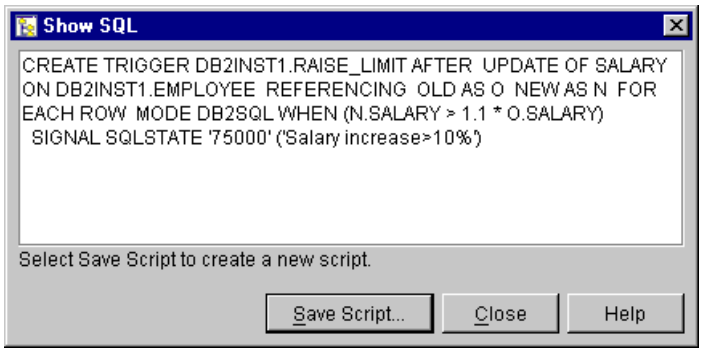

Figure 30. Creating a trigger from the Control Center (4)

6. When you are done viewing the SQL, choose **Close**, then **OK**, on the **Triggered Action** tab. If you receive no error message, the trigger has been successfully created, and you may view it in the Control Center.

7.8 Stored procedure conversion

A stored procedure resides on a database server, executes, and accesses the database locally to return information to client applications. Using stored procedures allows a client application to pass control to a stored procedure on the database server. This allows the stored procedure to perform intermediate processing on the database server, without transmitting unnecessary data across the network. Only those records that are actually required at the client need to be transmitted. This can result in reduced network traffic and better overall performance.

A stored procedure also saves the overhead of having a remote application pass multiple SQL statements to a database on a server. With a single statement, a client application can call the stored procedure, which then performs the database access work and returns the results to the client application. The more SQL statements that are grouped together for execution in the stored procedure, the larger the savings resulting from avoiding the overhead associated with network flows for each SQL statement when issued from the client.

Prior to DB2 UDB Version 7.1, you needed to develop stored procedures using external programming language such as C, whereas Sybase has allowed you to write stored procedures using the T-SQL language. In DB2 UDB Version 7.1, you can write stored procedures whose procedural logic is contained in the `CREATE PROCEDURE` statement. This type of stored procedure is called an SQL procedure. In this section, we discuss the conversion from T-SQL stored procedure in Sybase to SQL procedures in DB2 UDB.

7.8.1 Setting the environment to build SQL procedures in DB2 UDB

When you execute a `CREATE PROCEDURE` statement to build an SQL procedure, DB2 UDB generates a C program in which the SQL statements of the procedure body are embedded, and then pre-compiles it (compiles it into a executable file in the background). Therefore, you must install a supported C or C++ compiler on the DB2 UDB server.

Supported Compiler

For the AIX platform, either of the following compilers must be installed:

- IBM C for AIX Version 3.6.6
- IBM C Set++ for AIX Version 3.6.6
- IBM VisualAge C++ for AIX Version 4.0

Setting up environment variables

You can provide the value of PATH, INCLUDE, and LIBPATH environment variables for the compiler on the DB2 UDB server. You can create a batch file to set those environment variables and specify it using the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable, as follows:

```
db2set DB2_SQLROUTINE_COMPILER_PATH=Batch_File
```

In this statement, *Batch_File* is the full path name for the batch file setting environment variables.

If you do not set the DB2_SQLROUTINE_COMPILER_PATH DB2 registry variable, the default file DB2HOME/function/routine/sr_cpath is used (DB2HOME is the home directory of instance owner for DB2 UDB). This default file is generated automatically by DB2 UDB.

Compiler options

If you do not specify the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable, the default compiler command is as follows:

```
xlC -H512 -T512 -I$HOME/sqllib/include SQLROUTINE_FILENAME.c
-bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY
-o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2
```

If you want to change it, specify the new compiler command by using the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable on the server. In our test environment, we needed to set the DB2_SQLROUTINE_COMPILE_COMMAND DB2 registry variable as follows, because we used IBM C for AIX Version 3.6.6:

```
xlc -H512 -T512 -I$HOME/sqllib/include SQLROUTINE_FILENAME.c
-bE:SQLROUTINE_FILENAME.exp -e SQLROUTINE_ENTRY"
-o SQLROUTINE_FILENAME -L$HOME/sqllib/lib -lc -ldb2
```

Precompile and bind options

As already described, executing a CREATE PROCEDURE statement generates an embedded SQL program and pre-compiles/compiles it. You can provide precompile and bind options to that process using the DB2_SQLROUTINE_PREPOPTS DB2 registry variable on the DB2 UDB server, as in the following example:

```
db2set "DB2_SQLROUTINE_PREPOPTS=BLOCKING ALL ISOLATION UR"
```

The following options can be set for the DB2_SQLROUTINE_PREPOPTS DB2 registry variable:

- BLOCKING {UNAMIBIG | ALL | NO}
- DATETIME {DEF | USA | EUR | ISO | JIS | LOC }

- DEGREE {1 | degree-parallelism | ANY }
- EXPLAIN { NO | YES | ALL }
- EXPLAINSNAP { NO | YES | ALL }
- INSERT { DEF | BUF }
- ISOLATION { CS | RR | US | RS | NC }
- QUERYOPT optimization-level
- SYNCPOINT { ONEPHASE | TWOPHASE | NONE }

When you change the `DB2_SQLROUTINE_PREPOPTS` DB2 registry variable on the DB2 UDB server, you do not need to restart the DB2 server to make this change available.

For detailed information about these options, consult the description of the PRECOMPILE PROGRAM in the *DB2 UDB Command Reference*, SC09-2950.

7.8.2 Converting stored procedures from Sybase to DB2

This section discusses the conversion of stored procedures including the following topics:

1. Error handling in procedure
2. Simple procedure with 1 result set
3. Procedure with multiple result sets
4. Procedure with a single row
5. Nested procedures
6. Procedure with save points
7. Procedure that treats `DATETIME` data
8. Procedure with a `SET ROWCOUNT` statement
9. Procedure with temporary table

The first six sections discuss basic topics of conversion, and the last three sections discuss the stored procedure conversion which we performed for our customer application.

7.8.2.1 Error handling

Sybase has the `@@error` and the `@@sqlstatus` global variables for error handling. The `@@error` global variable has the error status after each execution of any SQL statements, and the `@@sqlstatus` global variable has the

status from the last FETCH statement. Its value is 0 (success), 1 (error), or 2 (no more data).

DB2 UDB has SQLSTATE and SQLCODE for error handling. The error handling model of the DB2 UDB's SQL stored procedure language is based on exception handling. An error not handled in the stored procedure makes it leave immediately before being able to check the SQLSTATE or the SQLCODE in the stored procedure, expecting the client to handle it.

For example, if the insert fails (for any reason), the DB2 stored procedure as it is translated here, will exit just after the INSERT statement not allowing you to set the ok variable based on the SQLSCODE.

- Original Sybase Stored Procedure:

```
create proc x(@mydate datetime,@ok integer out)
begin
   insert into tab(dt) values(@mydate)
   if(@@error = 0)         --- check @@error global variable
      select @ok=0;
   else
      select @ok=1;
end
```

- Direct translation to DB2 UDB

```
CREATE PROCEDURE X(IN mydate TIMESTAMP, OUT ok INTEGER)
LANGUAGE SQL
BEGIN
   DECLARE SQLCODE INTEGER DEFAULT 0;
   INSERT INTO tab(dt) VALUES(mydate); -- will exit here if fails
   IF(SQLCODE = 0) THEN    --- have an intention to check SQLCODE
      SET ok=0;
   ELSE
      SET ok=1;
   END IF;
END
```

Declaring error handler

To simulate the behavior of the @@error variable, the CONTINUE handler that updates your own error variable will have to be defined. This will prevent the stored procedure to exit prematurely.

The following stored procedure will behave like expected. If the insert fails, the declared error handler will be invoked and the variable for the SQLCODE will be set, then control will be returned to the statement that follows the INSERT statement that raised the exception.

```
CREATE PROCEDURE X(IN mydate TIMESTAMP, OUT ok INTEGER)
LANGUAGE SQL
BEGIN
   DECLARE SQLCODE INTEGER DEFAULT 0;                                    (1)
   DECLARE var_SQLCODE INTEGER DEFAULT 0;                                (2)
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND  (3)
      SET var_SQLCODE = SQLCODE;                                         (4)
   INSERT INTO tab(dt) VALUES(mydate);                                   (5)
   IF(var_SQLCODE <> 0) THEN                                             (6)
      SET ok=0;
   ELSE
      SET ok=1;
   END IF;
END
```

1. This statement is a declaration of variables for SQLCODE.
2. This statement is a declaration of variables to save SQLCODE.
3. This statement is a declaration of the handler for SQL exceptions. If an SQL exception occurs, the next statement (4) is executed and this procedure continues to the next step.
4. This statement is executed when SQL exception occurs. If you want to execute multiple statements here, you can compound them using the `BEGIN` and `END` statement.
5. This `INSERT` statement is executed.
6. This statement checks the value of SQLCODE, and sets the `ok` variable.

In this example, the variable to save the SQLCODE (`var_SQLCODE`) is initialized with the value 0 when it is declared, and it will be updated if the error handler is invoked.

This example has only one `INSERT` statement. If you want to execute more than one SQL statement and check the SQLCODE each time in the procedure, you need to initialize the user variable of the SQLCODE (`var_SQLCODE` in this example) with 0 before executing each SQL statement since the error handler is not invoked when an SQL statement succeeds. For example, assuming you have two `INSERT` statements in this stored procedure and the first statement fails but the second succeeds, the error handler would be invoked when the first statement fails and it would set a negative SQLCODE to the `var_SQLCODE` variable. However, the variable would keep the negative value even after the second statement succeeds, unless you initialize the variable with the zero value.

Note

To check the SQLCODE, you should initialize the user variable of SQLCODE with 0 before executing each SQL in a stored procedure.

RAISERROR command and SIGNAL statement

Sybase has the RAISERROR command for error handling. DB2 UDB has the SIGNAL statement, which features the same functionality. See the following example:

In Sybase:

```
create procedure showtable_sp @tabname varchar(18)
as
if not exists (select name from sysobjects where name = @tabname)  (1)
    begin
      raiserror 99999 "Table %1! not found.",@tabname              (2)
    end
else
    begin
      select name, type, crdate from  sysobjects where name = @tabname
    end
```

1. Here, we are checking the existence of the specified table.
2. If the table does not exist, the RAISERROR command is executed and return an error code to the caller program.

In DB2 UDB:

```
CREATE PROCEDURE showtable_sp(IN v_tabname VARCHAR(18))
RESULT SETS 1 LANGUAGE SQL
BEGIN
   DECLARE v_tmp_var CHAR(10);
   DECLARE v_temp_SQLCODE INT DEFAULT 0;
   DECLARE SQLCODE INT;
   DECLARE v_message varchar(128);
   IF NOT EXISTS(                                                  (1)
    SELECT name FROM sysibm.systables WHERE name = v_tabname) THEN
      set v_message = 'Table '||v_tabname||'! not found.';
      SIGNAL SQLSTATE '99999' SET MESSAGE_TEXT = v_message;        (2)
   ELSE
      BEGIN
         DECLARE SCW_Cur1 CURSOR WITH HOLD WITH RETURN
            FOR SELECT name, type, ctime
               FROM sysibm.systables
               WHERE name = v_tabname;
         DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
```

```
                                                                    (3)
            SET v_temp_SQLCODE = SQLCODE;

          OPEN SCW_Cur1;
        END;
      END IF;
    END
```

1. Here, we are checking the existence of the specified table.
2. If the table does not exist, the SIGNAL statement is executed, returning an error code and a message to the caller program. Note that this procedure does not have an error handler declaration in the same BEGIN - END block as the SIGNAL statement. If an error handler is declared in the same BEGIN - END block as the SIGNAL statement, the SIGNAL statement invokes the error handler and you need to issue the RESIGNAL statement in the error handler to return the specified error code and message. For more details about SIGNAL and RESIGNAL statements, see the *SQL Reference*, SC09-2974.
3. After checking the existence, you need to describe an error handler for error handling of other statements (the OPEN statement in this example) although this example does not have an error check routine here to make the example simple. You can declare one error handler for each BEGIN ... END block.

7.8.2.2 Using parameters

To send and receive data between the caller and the stored procedure, you can use parameters for a stored procedure. Parameters are defined in the CREATE PROCEDURE statement as the following:

In Sybase:

```
CREATE PROCEDURE showtable_sp @tabname VARCHAR(18) ...
```

In DB2:

```
CREATE PROCEDURE showtable_sp(IN v_tabname VARCHAR(18))...
```

If a stored procedure with parameters has been created, you need to specify the parameter in the CALL statement to execute the procedure in DB2 UDB.

In Sybase you can use the EXEC statement to execute a stored procedure having parameters. You can execute the EXEC statement without specifying parameter if the parameter definition in the CREATE PROCEDURE statement defines the default value. For example, if you have a stored procedure defined as follows:

```
CREATE PROCEDURE proc1 @var1 CHAR(1)= 'A' ...
```

Then the following two statements have the same meaning:

```
EXEC proc1
EXEC proc1 'A'
```

In DB2, you cannot omit the parameter from the CALL statement if the stored procedure has a parameter. Here is a simple example:

```
CALL proc1('A')
```

In DB2 UDB, you can change values of the parameters declared as OUT parameters, but you cannot change values of the parameters declared as IN parameters. If you want to change the values of the variables declared as IN parameters, you should specify the INOUT parameter option. In Sybase, you can change parameter variables of stored procedures if the parameter is not explicitly defined as an OUT parameter.

When you want to set a value to a variable in a stored procedure, you need to describe a statement like '`select @var = value`' in Sybase and '`set var = value`' in DB2 UDB.

7.8.2.3 Conversion of a simple procedure with one result set

This example shows a simple stored procedure that executes a SELECT statement and returns an answer set.

Source in Sybase:

```
create procedure prc_read_table
as
   declare @ret_code int
   select @ret_code = 0
   select * from table01
   if @@error != 0
      select @ret_code = @@error
   return @ret_code
```

To DB2 UDB:

```
CREATE PROCEDURE prc_read_table()
RESULT SET 1 LANGUAGE SQL                                          (1)
BEGIN                                                              (2)
   DECLARE SQLCODE INTEGER default 0;
   DECLARE var_SQLCODE INTEGER default 0;
   DECLARE Cur1 CURSOR WITH HOLD WITH RETURN                       (3)
      FOR SELECT  *
         FROM table01;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      SET var_SQLCODE = SQLCODE;
```

```
    OPEN Cur1;                                                    (4)
    RETURN(var_SQLCODE);
END
```

1. A `LANGUAGE SQL` statement must be specified for an SQL stored procedure. The `RESULT SET` statement exists for the family compatibility.
2. The SQL stored procedure body has to be enclosed by `BEGIN` and `END`.
3. You must have a `WITH RETURN` option in order to pass a result set back from the cursor.
4. You need to open the cursor that was declared by a `DECLARE CURSOR` statement to return a result set.

As in the example above, you need to declare and open a cursor, and then leave it open to return a result set to the caller.

In DB2 UDB, if want to execute this stored procedure in your application program, you need to use ODBC/CLI/JDBC interface. If you use embedded SQL interface, you cannot get the result set from the stored procedure; however, you can execute CLI statements in your embedded SQL program to get the result set from the stored procedure. The example is shown in 7.9.6, “Executing a stored procedure” on page 207.

You can also get the result set by executing the `CALL` statement from the DB2 command line processor.

Within a `BEGIN ... END` block of an SQL stored procedure in DB2 UDB Version 7.1, you must put statements as the following order:

- Variables declaration
- `DECLARE CURSOR` statements
- Error handler declaration
- SQL procedure statements

If you put `DECLARE CURSOR` statements after a handler declaration, you will get the error ‘`SQL0104N An unexpected token "<cursor declaration>" was found following "".`’.

7.8.2.4 Conversion of procedure with multiple result sets

This example shows a stored procedure that executes two `SELECT` statements and returns two answer sets.

In Sybase:

```
create procedure prc_read_tables(@srvr_nm varchar(40))
```

```
as
   declare @ret_code int
   select @ret_code = 0
   select * from table01 where col01 = @srvr_nm                    (1)
   if @@error != 0
      select @ret_code = @@error
   select * from table02 where col01 = @srvr_nm                    (2)
   if @@error != 0
      select @ret_code = @@error
   return @ret_code                                                (3)
```

1. Executes SELECT statements for the first result set.
2. Executes SELECT statements for the second result set.
3. Returns an error code.

In DB2 UDB:

```
create procedure prc_read_tables(v_srvr_nm varchar(40))
result set 2 language SQL
begin
   declare SQLCODE INTEGER default 0;
   declare var_SQLCODE INTEGER default 0;
   declare ret_code INTEGER default 0;

   DECLARE C1 cursor with hold with return                         (1)
      FOR select * from table01 where col01 = v_srvr_nm;
   DECLARE C2 cursor with hold with return                         (2)
      FOR select * from table02 where col01 = v_srvr_nm;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      SET var_SQLCODE = SQLCODE;
   open C1;                                                        (3)
   if(var_SQLCODE <> 0) then
      set ret_code = var_SQLCODE;
   end if;
   set var_SQLCODE = 0;                                            (4)
   open C2;                                                        (5)
   if(var_SQLCODE <> 0) then
      set ret_code = var_SQLCODE;
   end if;
   return(ret_code);                                               (6)
END
```

1. Declares cursors for the first result set.
2. Declares cursors for the second result set.
3. Opens declared cursor for the first result set.
4. Sets the user variable of SQLCODE to 0 before executing SQL statement.

5. Opens declared cursor for the second result set.
6. Returns an error code.

Note that two cursors are declared, opened, and left open.

7.8.2.5 Conversion of procedure with a single row

Both Sybase and DB2 have the SELECT statement to retrieve a value from the table and save it into a variable. However, the behavior is different when the SELECT statement returns more than one value.

This is an example of the stored procedure using the SELECT statement in Sybase:

```
create procedure prc_single(@col01 char(5),@col02 char(5) out)
as
   select @col02 = col02 from table01 where col01 = @col01
```

If the result set of this statement has only 1 row, you can convert the stored procedure as following in DB2 UDB:

```
create procedure prc_single(IN v_col01 char(5),OUT v_col02 char(5))
language sql
BEGIN
   SELECT col02 into v_col02 from table01 where col01=v_col01;
END
```

If the result set has more than 1 row, Sybase will set the last value in the result set to the variable @col02; however, in DB2 UDB, the SELECT statement will fail with the error code SQL0811N, because the SELECT...INTO statement allows only one row to be returned.

If you want to create a stored procedure which has the same behavior as the Sybase stored procedure, you need to use a cursor and return the last fetched data to the caller. See the following stored procedure:

```
create procedure prc_single2(IN v_col01 char(5),OUT v_col02 char(5))
language sql
BEGIN
   DECLARE v_col01 char(5);
   DECLARE SQLCODE INTEGER default 0;
   DECLARE v_SQLCODE INTEGER default 0;
   DECLARE c1 CURSOR             -- Declare Cursor
      FOR SELECT col02 FROM table01 where col01=v_col01;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      set v_SQLCODE = SQLCODE;   -- For error handling
   OPEN c1;                      -- Open Cursor
   fetch_loop:
```

```
   LOOP
      FETCH c1 INTO v_col02;      -- Fetch each 1 row
      IF(v_SQLCODE = 100) then    -- Check the end of data
         LEAVE fetch_loop;
      END IF;
   END LOOP;
   CLOSE c1;                      -- Close Cursor
END
```

When the last FETCH statement is executed, the variable 'v_col02' is set to the last value in the result set.

7.8.2.6 Conversion of a procedure which calls another procedure

Both Sybase and DB2 UDB support nested stored procedures. There are some differences in the usages of cursors between Sybase and DB2 UDB. You can use cursors to manipulate result sets of SELECT statements both in Sybase and DB2 UDB. In Sybase, when a cursor is defined in a stored procedure or an user program, the called stored procedure or trigger can access the cursor. See the example in Figure 31.

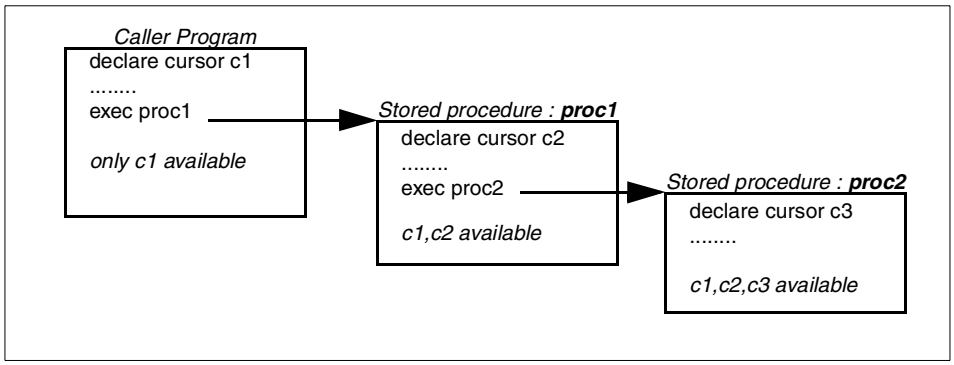

Figure 31. Cursor scopes in Sybase

The caller program can access the cursor c1 that is declared in the program, but cannot access the other cursors (c2 and c3). The Stored procedure proc1 can access the cursors (c2 and c1) that are declared in the same stored procedure and the caller program, but cannot access the cursor c3. The stored procedure proc2 can access the cursors c3, c2, and c1.

In DB2 UDB, the called stored procedure cannot access the cursor that was defined in the caller program. But the caller program or stored procedure can access the cursor that was defined in the called stored procedure. See the examples shown in Figure 32.

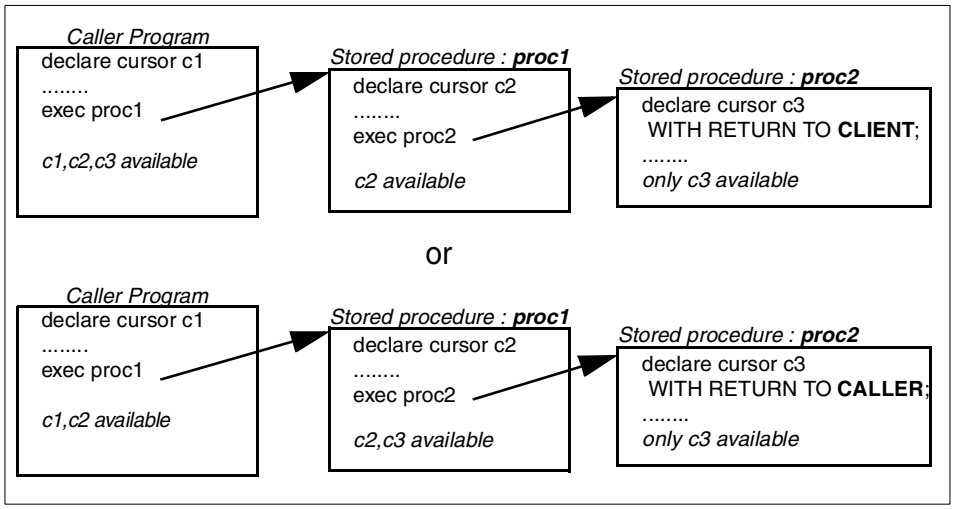

Figure 32. How to get cursor from stored procedure in DB2 UDB

The caller program can access the cursor `c1` and `c2`. The cursor `c3` is available to the caller program when the cursor is defined with `RETURN TO CLIENT` option in the called stored procedure `proc2`. The cursor `c3` is available in the procedure `proc1` when the cursor is defined with `RETURN TO CALLER` option in the called stored procedure `proc2`. The cursor `c2` can be accessed by the caller program with either `RETURN TO CLIENT` or `RETURN TO CALLER` option.

The caller stored procedure `proc1` can access the cursor `c3` declared and opened in the called stored procedure `proc2` with a result set locator. You have to use the `ALLOCATE CURSOR` and `ASSOCIATE RESULT SET LOCATOR` statements to access the cursor which open in the called stored procedure as the following example:

```
DECLARE loc1 RESULT_SET_LOCATOR VARING;   --Declare a result set locator
.....
CALL proc2;
ASSOCIATE RESULT SET LOCATORS loc1 WITH PROCEDURE proc2;
                              --Associate the locator to the procedure
ALLOCATE c3 CURSOR FOR RESULT SET loc1;
                                 --Allocate the cursor with the locator
FETCH c3 INTO var1;               --Fetch the cursor
```

Simulating the Sybase behavior

If you need to access the cursor that is defined in the caller program (or stored procedure) from the called stored procedure like you do in Sybase, you cannot meet the requirement with those statements such as `ALLOCATE CURSOR`

and ASSOCIATE RESULT SET LOCATOR. Instead you can use a declared temporary table to implement the same functionality. When you define a declared temporary table in the caller program, the called stored procedure can access the declared temporary table. See Figure 33.

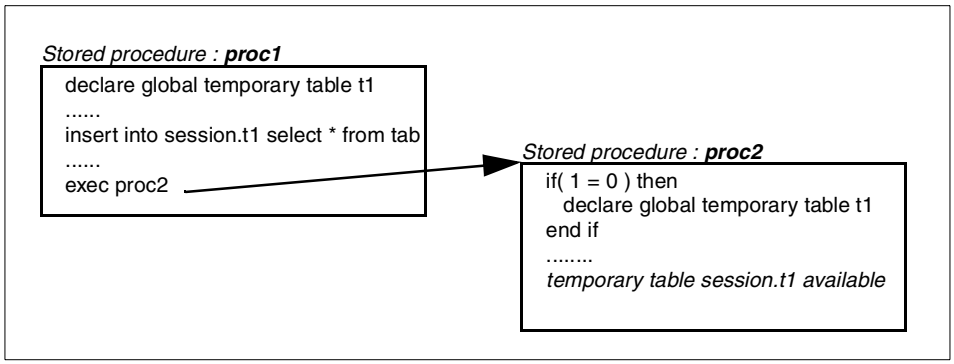

Figure 33. Simulate Sybase cursor in DB2 UDB

In this example, the result set of the SELECT statement in the stored procedure proc1 is passed to the called stored procedure proc2 using a declared temporary table.

Note that the stored procedure proc2 has a dummy DECLARE GLOBAL TEMPORARY TABLE statement to avoid getting an error when creating this procedure. If you do not put in this statement, you will get an error when executing an SQL statement accessing the temporary table. In DB2 UDB Version 7.1, if you want to access the declared temporary table that are declared by the other program or procedure, you need to describe like this.

7.8.2.7 Conversion of procedure with save points

Between Sybase and DB2 UDB, the transaction models are different. Sybase supports nested transactions whereas DB2 UDB does not. For example, if you have an application program that calls a stored procedure, the program can start a transaction that calls the stored procedure, and a new transaction can be started by the stored procedure within the transaction that the caller program has started. Then you can use the COMMIT statement for the transaction that is started in the stored procedure.

In DB2 UDB, if you issue the COMMIT statement in the called stored procedure, all changes made in the stored procedure and the caller program will be committed.

Both Sybase and DB2 UDB support save points to control transactions. See the Sybase example shown in Figure 34. This example does not have a save point.

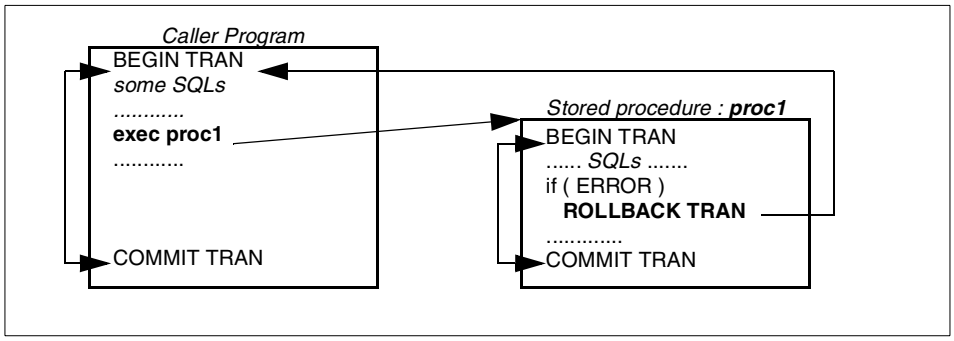

Figure 34. Case 1: No save point in a Sybase stored procedure

In this example, the `ROLLBACK` statement in the stored procedure `proc1` will roll back all the changes made in the transaction started by the stored procedure, as well as all the changes made in the transaction started in the caller program. To roll back only the transaction started by the stored procedure, you should use a save point in the stored procedure. See the following Sybase example shown in Figure 35.

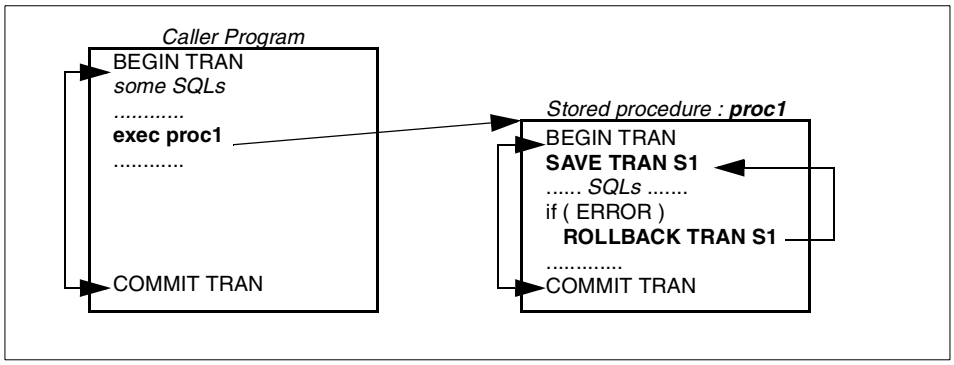

Figure 35. Case 2: A save point in a Sybase stored procedure

In this example, the `ROLLBACK` statement in the stored procedure `proc1` can roll back only the transaction started in this procedure by specifying the save point name.

If a `BEGIN TRAN` statement is not used in the caller program, each SQL procedure will be executed with the auto-commit mode in the caller program.

In that case, the save point does not need to be put in the called stored procedure, because either a COMMIT statement or a ROLLBACK statement will affect the transaction within the stored procedure only.

In DB2 UDB, you can also use save points in a stored procedure for the same purpose as Sybase. In DB2 UDB, a transaction starts implicitly when an SQL statement (SELECT, UPDATE, DELETE, and so on) is executed. See the DB2 example shown in Figure 36.

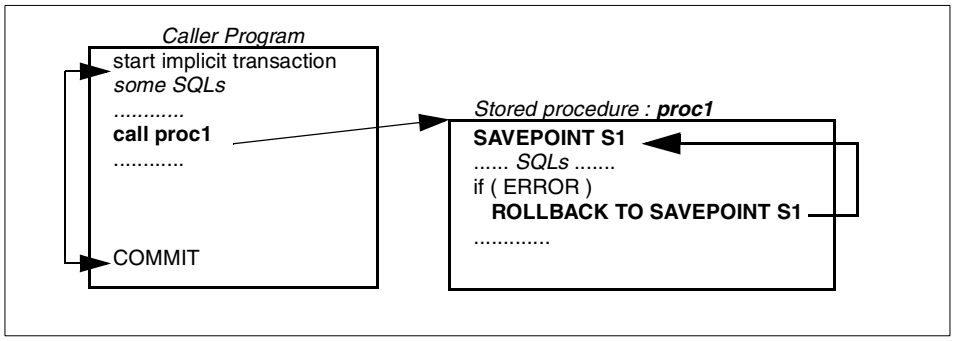

Figure 36. A save point in a DB2 UDB stored procedure

This example has a program calling a stored procedure. In this example, if you want to roll back the SQL statements executed within the stored procedure only, you should set a save point (in this example, S1). If the called stored procedure does not have a save point, executing the ROLLBACK statement will roll back all the changes both in the caller program and the stored procedure.

As described before, DB2 does not support nested transaction. Thus, we recommend to put the COMMIT statement in the caller program, not in the called stored procedure. If you issue the COMMIT statement at the end of this stored procedure, the commit statement will commit not only the changes in the caller program but also the changes in the stored procedure. That means the caller program cannot control the transaction which is started by itself. To avoid this situation, you should commit or roll back transactions in the caller program. In stored procedures, use the SAVEPOINT and the ROLLBACK SAVEPOINT statements.

We discussed save points in 7.5, “Save point” on page 147. The following example shows the use of save point in Sybase:

```
create proc prc_sptest()
as
```

```
   declare @saverr int;
   select @saverr = 0;
begin tran tr_sptest                                                    (1)
save tran s_sptest                                                      (2)
update table01 set col02='DDDDD' where col01='00002';
select @saverr = @@error;
if @saverr != 0                                                         (3)
   rollback tran s_sptest                                               (4)
commit tran                                                             (5)
return @saverr;                                                         (6)
```

This stored procedure start a transaction and have the save point `s_sptest`. If the `UPDATE` statement fails, this procedure rolls back to the save point `s_sptest` and then commits the transaction `tr_sptest`.

1. "`begin tran`" statement starts a transaction that is named "`tr_sptest`"
2. "`save tran`" statement sets a save point named "`s_sptest`"
3. This statement checks the return code from the `UPDATE` statement. If the code is not 0, `ROLLBACK` statement (4) will be executed.
4. "`rollback tran`" statement rolls back to the save point "`s_sptest`".
5. "`commit tran`" statement ends the transaction that was started by the "begin tran" statement (1).
6. This procedure returns an error code to the caller program or procedure. The caller program or procedure retrieves the error code as follows:

   ```
   exec @rc_code = prc_sptest;
   ```

   This example sets the error code from the procedure `prc_sptest` to a local variable `@rc_code`.

We can convert this stored procedure to DB2 UDB as the following example:

```
create procedure prc_sptest()
language sql
begin                                                                   (1)
   declare saverr int;
   declare SQLCODE int;
   declare continue handler for sqlexception,sqlwarning,not found       (2)
      set saverr = SQLCODE;
   set saverr = 0;
   savepoint s_sptest on rollback retain cursors;                       (3)
      update table01 set col02='DDDDD' where col01='00002';
   if (saverr <> 0) then                                                (4)
      rollback to savepoint s_sptest;
   end if;
   return saverr;                                                       (5)
```

```
end
```

1. This `begin` statement means only the beginning of an SQL block. This does not mean beginning a transaction. In DB2 UDB, a transaction starts automatically when the `UPDATE` statement is executed in this procedure.
2. If an error occurs, this statement sets `saverr` variable to the value of SQLCODE.
3. This statement sets a save point named `s_sptest`.
4. This statement checks the error code. If the code is not 0, this executes the `ROLLBACK TO SAVEPOINT` statement.
5. This procedure returns an error code to the caller program. If the caller is a stored procedure, you can get the error code from the called procedure as follows:

```
CALL prc_sptest();
GET DIAGNOSTICS <variable> = RETURN_STATUS;
```

See the details in 7.9.6, “Executing a stored procedure” on page 207 for the method to get the error code when you call the stored procedure from a embedded SQL program.

7.8.2.8 Conversion example that treats DATETIME data

In our customer database, we had an example treating `DATETIME` data. Sybase has a `DATETIME` data type that includes year, month, day, time, and millisecond. DB2 UDB has a `TIMESTAMP` data type that includes year, month, day, time, and microsecond. That `DATETIME` data type in Sybase can be converted to a `TIMESTAMP` data type in DB2 UDB; however, their formats are different as discussed in 4.2.3, “Datetime data type” on page 45 and we needed to take it as a consideration.

Here is the source stored procedure in Sybase. The data type of the column `col03` is `DATETIME`:

```
create procedure dateproc1 (
   @var_col01 char(10) = "%"                                          (1)
)as
   select  col01, count(col01), col02, convert(char (12), col03)
   from table01
   where col01 like @var_col01
   group by col01, col02, convert(char (12), col03),                  (2)
   datepart(weekday,col03)
   order by col01,datepart(weekday,col03)
```

1. The input parameter is defined with the default value ‘`%`’.

2. The CONVERT function extracts the year, month, and day part of the DATETIME data.

The following code is the converted stored procedure for DB2:

```
CREATE PROCEDURE dateproc1(IN var_col01 CHAR(10))
RESULT SETS 1 LANGUAGE SQL
BEGIN NOT ATOMIC
   DECLARE v_temp_SQLCODE INT DEFAULT 0;                          (1)
   DECLARE SQLCODE INT;
   DECLARE cur1 CURSOR WITH HOLD WITH RETURN                      (2)
      FOR SELECT col01,count(col01),col02,CHAR(date(col03))       (3)
       FROM table01
       WHERE col01 LIKE var_col01
       GROUP BY col01,col02,CHAR(date(col03)),DAYOFWEEK(col03)
       ORDER BY col01,DAYOFWEEK(col03);
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
         SET v_temp_SQLCODE = SQLCODE;
   OPEN cur1;                                                     (4)
END
```

1. These variables are declared for error handling.
2. Declaration of a cursor to return a result set. A "WITH HOLD" option will keep the cursor open after COMMIT is issued. A "WITH RETURN" is also needed to return a result set.
3. In Sybase, CONVERT(CHAR(12),*datetime*) returns year, month, and day in the format like 'Aug 28 2000'. In DB2 UDB, you can use the DATE function to return year, month, and day. Note that the format will be like '08/28/2000'.
4. To return a result set, the cursor has to be opened.

Note that with in a begin-end block, a cursor declaration must be done before the error handler declaration as described in "Conversion of a simple procedure with one result set" on page 175.

7.8.2.9 Conversion example of SET ROWCOUNT

The next example has also been found in our customer database. This example uses the SET ROWCOUNT statement which limits a number of rows that the DELETE statement processes. The reason for limiting a number for deleted rows is to prevent the transaction log space from getting full. In this example, the DELETE statement deleting 1000 rows is repeatedly executed with the auto-commit mode until all the rows which meets the specified condition is deleted. Here is the source Sybase stored procedure:

```
create procedure procdel
as
```

```
declare  @ret_day          int,
         @del_count        int

select @del_count = 0

select @ret_day = convert(int,col01) * -1                          (1)
   FROM dbo.table02
   where col02 = 'RET_DATE'
   and col03 = @@servername

set rowcount 1000                                                  (2)
declare @save_count int
set nocount on

while (1=1)
   begin
      DELETE FROM dbo.table03
         WHERE col01 < dateadd(dd,@ret_day,getdate())              (3)
      select @save_count = @@rowcount                              (4)
      select @del_count = @del_count + @save_count
      if @save_count < 1000
         break
   end

select 'Daily rows deleted',@del_count                             (5)
```

1. This SELECT statement gets the value for the retention date.
2. This statement sets the maximum number of the rows that are processed by the following SQL statement.
3. This DELETE statement deletes 1000 rows at a time as a maximum.
4. In Sybase, you can input a value to a variable using the SELECT statement like this. The @@rowcount shows a number of rows that are deleted by the DELETE statement (3).
5. The message and the number of the deleted rows are returned to the caller by the SELECT statement.

DB2 UDB does not have an equivalent feature to control the number of the rows being deleted. In your migration project from Sybase to DB2 UDB, if you have a Sybase stored procedure deleting many rows and using the SET ROWCOUNT statement to prevent the transaction log space getting full, executing multiple DELETE statements with a different condition, and increasing the transaction log space would be a practical method to convert such a procedure. See the following converted example of the DB2 stored procedure:

```
CREATE PROCEDURE procdel()
 RESULT SETS 1 LANGUAGE SQL
BEGIN
   DECLARE v_temp_SQLCODE INT DEFAULT 0;
   DECLARE SQLCODE INT;
   DECLARE v_ret_day INT;
   DECLARE v_del_count INT;
   DECLARE v_rowcount INT;
   DECLARE cur1 CURSOR WITH HOLD WITH RETURN                          (1)
      FOR WITH temptable(col1, col2) AS (VALUES ('Daily rows deleted',
         V_del_count))
         SELECT  * FROM temptable;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      BEGIN NOT ATOMIC
         SET v_temp_SQLCODE = SQLCODE;
      END;
   SET v_del_count = 0;
   SET v_temp_SQLCODE = 0;
   SELECT CAST(col01 AS INT) * -1 INTO v_ret_day
      FROM table02
      WHERE col02 = 'RET_DATE' AND col03 = CURRENT SERVER;
   SET v_temp_SQLCODE = 0;
   DELETE FROM table03                                                (2)
      WHERE col01 <  (v_ret_day DAYS+(CURRENT DATE) -5);
   GET DIAGNOSTICS v_rowcount = ROW_COUNT;                            (3)
   SET v_del_count = v_del_count + v_rowcount;
   SET v_temp_SQLCODE = 0;
   DELETE FROM table03
      WHERE col01 <  (v_ret_day DAYS+(CURRENT DATE));
   GET DIAGNOSTICS v_rowcount = ROW_COUNT;
   SET v_del_count = v_del_count + v_rowcount;

   OPEN cur1;                                                         (4)
END
```

1. The SELECT statement returning the message and the value in Sybase can be converted into a cursor in DB2 UDB. Here the cursor is declared using the SELECT statement with the WITH clause. The cursor is opened by the OPEN statement.

   Note: You can also use stored procedure parameters to return values to the caller.

2. The DELETE statement is executed. In this example, the DELETE statement is executed twice.

3. "GET DIAGNOSTICS <variable>=ROW_COUNT" statement shows a number of rows that are processed by the DELETE statement. We have discussed this statement in 7.6.5, "The @@rowcount global variable" on page 154.

Since the source Sybase stored procedure does not have any error checking routines, we did not put any of them in the converted stored procedure either, although the error handler sets a negative value to the variable v_temp_SQLCODE if any error occurs. You can check the variable v_temp_SQLCODE if you want to add an error check routine.

If you really want to create a stored procedure which behaves in the same way as the source Sybase stored procedure, which use the SET ROWCOUNT statement and delete 1000 rows each time, you can use a cursor using the SELECT FOR UPDATE and execute DELETE CURRENT OF *Cursor* statements.

In the following procedure, we use a SELECT FOR UPDATE statement and a DELETE CURRENT OF *Cursor* statement to delete 1000 rows each time.

```
CREATE PROCEDURE procdel()
RESULT SETS 1 LANGUAGE SQL
BEGIN NOT ATOMIC
DECLARE v_temp_SQLCODE INT DEFAULT 0;
DECLARE SQLCODE INT;
   DECLARE v_ret_day INT;
   DECLARE v_del_count INT;
   DECLARE v_rowcount INT;
   DECLARE v_col01 INT;

   DECLARE cur1 CURSOR WITH HOLD WITH RETURN
      FOR WITH temptable(col1, col2) AS (VALUES ('Daily rows deleted',
         v_del_count))
         SELECT * FROM temptable;
   DECLARE cur2 CURSOR WITH HOLD                                    (1)
       FOR SELECT col01 FROM test03
           WHERE col01 <  (v_ret_day DAYS+(CURRENT TIMESTAMP))
           FOR UPDATE;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      BEGIN NOT ATOMIC
      SET v_temp_SQLCODE = SQLCODE;
      END;
   SET v_del_count = 0;
   SET v_temp_SQLCODE = 0;
   SELECT CAST(col01 AS INT)  *  -1 INTO v_ret_day
      FROM test02
      WHERE col02 = 'RET_DATE' AND col03 = CURRENT SERVER;
   SET v_temp_SQLCODE = 0;
```

```
   SET v_rowcount=0;
   OPEN cur2;
   L_Label:
   WHILE (1 = 1)
   DO
      FETCH cur2 INTO v_col01;                                    (2)
      DELETE FROM test03 WHERE current of cur2;                   (3)
      SET v_rowcount = v_rowcount + 1;                            (4)
      IF MOD(v_rowcount,1000) =0 THEN                             (5)
         COMMIT WORK;
      END IF;
      IF v_temp_SQLCODE <> 0 THEN
         SET v_del_count = v_rowcount;
         COMMIT WORK;
         Leave L_Lavel;
      END IF;
   END WHILE L_Label;
   OPEN cur1;
   COMMIT WORK;                                                   (6)
   CLOSE cur2;
END
```

1. In this SELECT FOR UPDATE statement, we need to use a WITH HOLD option to retain the cursor opened after a COMMIT statement.
2. The FETCH statement moves the position of the current cursor to the next row.
3. The DELETE statement deletes 1 row where the current cursor is placed.
4. The v_rowcount value counts a number of rows that are deleted.
5. For each 1000 rows deleted, the COMMIT statement is executed.

7.8.2.10 Conversion example of declared temporary tables

This example using a local temporary table is also from our customer database. As already described, you can create a temporary table using the DECLARE GLOBAL TEMPORARY TABLE statement in DB2 UDB.

This stored procedure returns a result set consisting of two columns. One is the words extracted from the column col01 of the table table04 and the other column is the sequential number beginning with 1. The column col01 of the table table04 stores multiple words delimited by comma in a row. This stored procedure picks up each word from the beginning using the CHARINDEX, SUBSTRING, and the STUFF function

Here is the source stored procedure in Sybase:

```
create procedure proctemp (
```

```
   @parm01     char(10)
)
as
declare @list varchar(255)
declare @tmp_list varchar(255)
declare @offset smallint
declare @v_word varchar(30)
declare @v_word_num smallint

create table #temptable (work char(10),word_num smallint)          (1)
select @v_word_num = 1

select @list = col01 from table04 where col02 = @parm01
   while (@list != '')
   begin
   select @offset = charindex(",", @list)                          (2)
   if @offset > 0
      begin
         select @v_word = substring(@list, 1, @offset-1)           (3)
         insert #temptable select @v_word, @v_word_num             (4)
         select @tmp_list = stuff(@list, 1, @offset, NULL)         (5)
         select @list = @tmp_list
         select @v_word_num = @v_word_num + 1
      end
   else
      begin
         insert #temptable select @list, @v_word_num               (6)
         select @list = ''
      end
   end

select * from #temptable                                           (7)
```

1. This statement creates a local temporary table named `#temptable`.
2. The CHARINDEX function returns the position of the first comma character. This function should be converted to the LOCATE or POSSTR function in DB2 UDB.
3. This statement picks up the first word using the SUBSTRING function. It should be converted to the SUBSTR function in DB2 UDB.
4. This statement inserts the extracted word and the sequential number into the local temporary table.
5. The word which has already been inserted into the local temporary table is eliminated from the list by this statement using the STUFF function. This function should be converted to a INSERT function in DB2 UDB.

6. The last word is inserted into the local temporary table.
7. The content of the local temporary table is returned to the caller.

The following code is converted stored procedure for DB2.

```
CREATE PROCEDURE proctemp(IN parm01 CHAR(10))
RESULT SETS 1 LANGUAGE SQL
BEGIN NOT ATOMIC
   DECLARE v_temp_SQLCODE INT DEFAULT 0;
   DECLARE SQLCODE INT;
   DECLARE v_list VARCHAR(255);
   DECLARE v_tmp_list VARCHAR(255);
   DECLARE v_offset SMALLINT;
   DECLARE v_word VARCHAR(30);
   DECLARE v_word_num SMALLINT;
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION,SQLWARNING,NOT FOUND
      SET v_temp_SQLCODE = SQLCODE;
   DECLARE GLOBAL TEMPORARY TABLE SESSION.temptable (              (1)
      word CHAR(10)        NOT NULL,
      word_num SMALLINT NOT NULL)
      NOT LOGGED WITH REPLACE ON COMMIT PRESERVE ROWS;
   SET v_word_num = 1;
   SET v_temp_SQLCODE = 0;
   SELECT col01 INTO v_list FROM table04 WHERE col02 = parm01;
   WHILE (v_list <> '')
   DO
      SET v_offset = LOCATE( ',', v_list) ;                        (2)
      IF v_offset >  0 THEN
         SET v_word = SUBSTR(v_list, 1, v_offset -  1);            (3)
         SET v_temp_SQLCODE = 0;
         INSERT INTO SESSION.temptable VALUES (v_word, v_word_num);
         VALUES(INSERT(v_list,1,v_offset,'')) INTO v_tmp_list;     (4)
         SET v_list = v_tmp_list;
         SET v_word_num = v_word_num + 1;
      ELSE
         SET v_temp_SQLCODE = 0;
         INSERT INTO SESSION.temptable VALUES (v_list, v_word_num);
         SET v_list = '';
      END IF;
   END WHILE;
   BEGIN
      DECLARE cur1 CURSOR WITH HOLD WITH RETURN                    (5)
         FOR SELECT * FROM SESSION.temptable;
      OPEN cur1;
   END;
```

1. Create a declared global temporary table. The WITH REPLACE option is used to override the existing declared global temporary table when you execute this stored procedure more than once with in a connection. The ON COMMIT PRESERVE ROWS option is used to keep data in the declared temporary table after COMMIT statement. The name of this temporary table is SESSION.temptable.
2. The CHARINDEX function in Sybase is converted into the LOCATE function in DB2.
3. The SUBSTRING function in Sybase is converted into the SUBSTR function in DB2.
4. The STUFF function in Sybase is converted into the INSERT function in DB2. Note that two single quotation without a space (’’) are used to specify NULL value whereas the STUFF function uses the string NULL in the Sybase stored procedure. We have discussed this difference in 7.3.1, “Compatible functions” on page 127.
5. A cursor needs to be declared using the DECLARE CURSOR statement to return a result set. As discussed in 7.8.2.3, “Conversion of a simple procedure with one result set” on page 175, you need to put the DECLARE CURSOR statement before the error handler declaration, and the other SQL statements have to be put after the error handler declaration within a BEGIN-END block. In this example, the cursor accesses the declared temporary table and the table has to have been declared when the cursor is declared. Thus, you need to put one more BEGIN ... END block in this procedure to put the cursor declaration.

In this example, the cursor is declared using a static SQL statement. If you want to use a dynamic SQL statement in a stored procedure, the order of statements needs to be as follows:

```
BEGIN
   DECLARE str_stmt
   .....
   DECLARE CURSOR C1 FOR STMT1;
   .....Declare error handler, Other statements
   .....
   SET str_stmt = 'SELECT statement';
   PREPARE STMT1 FROM str_stmt;
   OPEN C1;
END
```

7.8.3 DB2 Stored Procedure Builder

DB2 UDB comes with the Stored Procedure Builder (SPB), a graphical application to aid in rapid development of stored procedures. SPB can provide a single development environment of the stored procedure, which means SPB can be used to build stored procedures for the entire DB2 family ranging from the Windows workstation to System/390 whether those procedures are created on local or remote server.

SPB comes with design assistants that guide you through basic design patterns, help you create SQL queries, and estimate the performance cost of invoking a stored procedure. SPB also helps you in registering and testing stored procedures. Therefore, you can focus on creating your stored procedure logic rather than the details of registering, building, and installing stored procedures on a DB2 server.

On a Windows workstation, SPB can be started by clicking **Start** and selecting **Programs -> IBM DB2 -> Stored Procedure Builder**, clicking the **SPB** icon from the Control Center, or entering `db2spb` at a command prompt. SPB can be also started as an add-in tool from Microsoft Visual Basic or Microsoft Visual C++. On a UNIX workstation such as RS/6000, you can start SPB by entering `db2spb` at a command prompt, or clicking the **SPB** icon from the Control Center.

When you start SPB, you are prompted to specify the characteristics of a new project or open an old project. A project stores the database name to which the stored procedures connect to, userid, and password, and so on. Stored procedures are created in the project.

Once you open a project, you are ready to build a new stored procedure. Right-click on the **Stored Procedures** folder which can be seen under a database folder, and select **Insert**. You can specify whether the language of the procedure is Java or SQL, as shown in Figure 37.

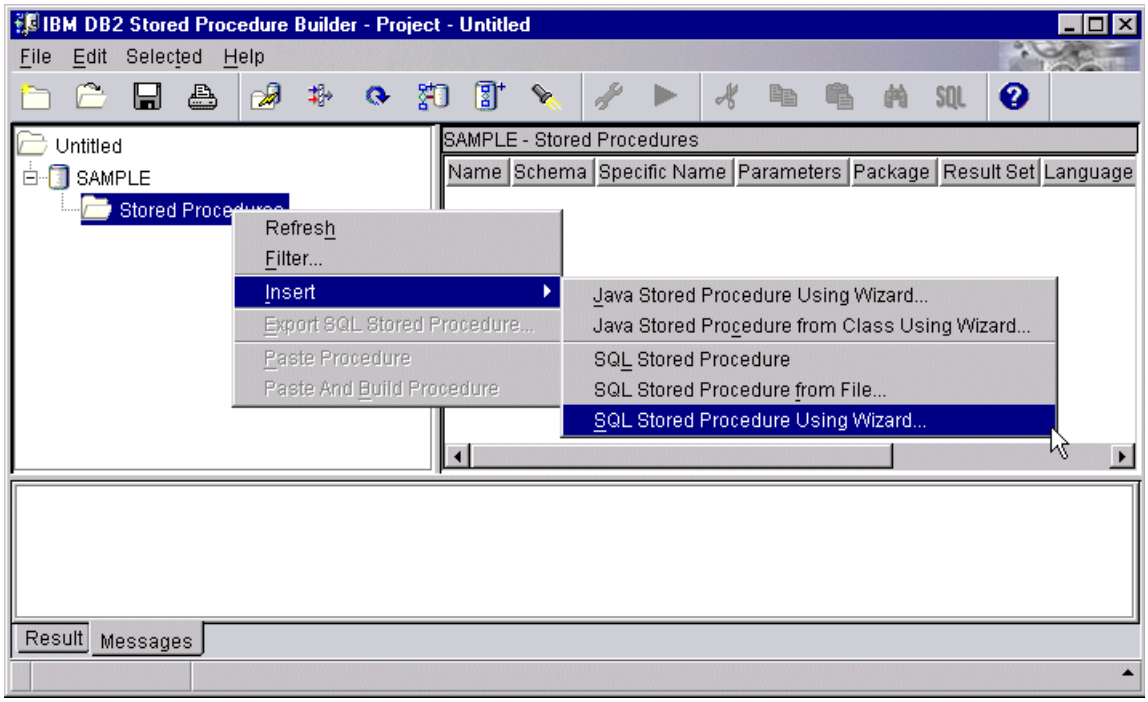

Figure 37. Stored Procedure Builder: inserting a new procedure

In our example, we are creating an SQL stored procedure using the wizard. You should just enter the prompted information. SPB requests you to enter the procedure name, SQL statements, input/output parameters, and so on. Figure 38 shows the page specifying SQL statements. You can directly enter the SQL statements, or use the SQL Assistant, which is a wizard to create SQL statements.

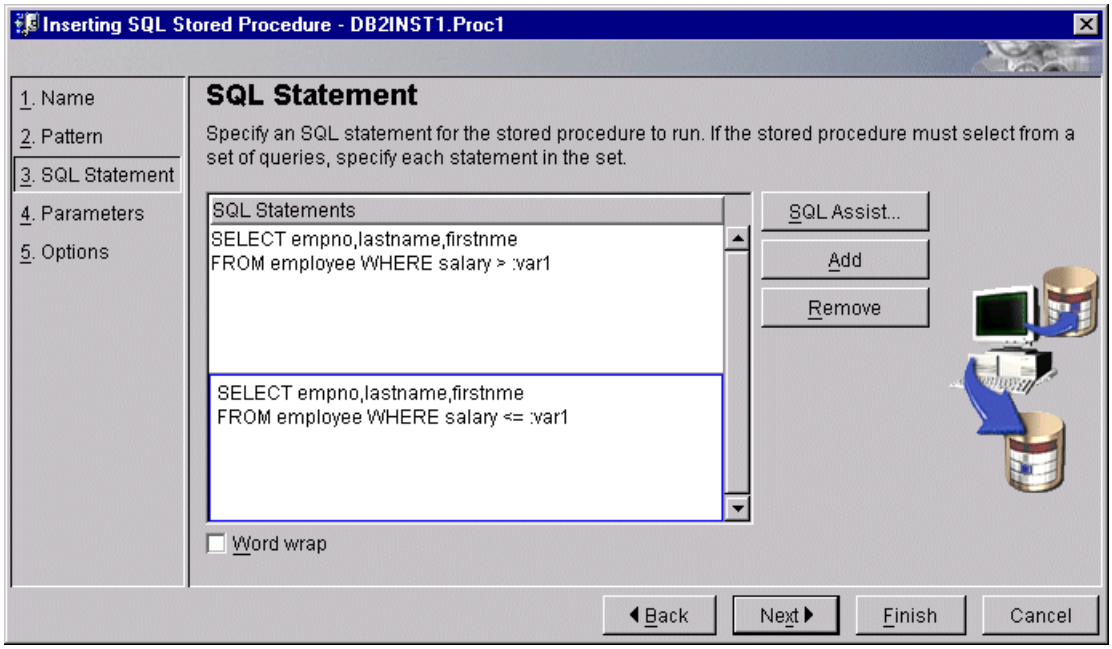

Figure 38. Stored Procedure Builder: using the wizard

When you have entered all prompted information, click the **Finish** button, and SPB generates the stored procedure code. Figure 39 shows the generated SQL stored procedure.

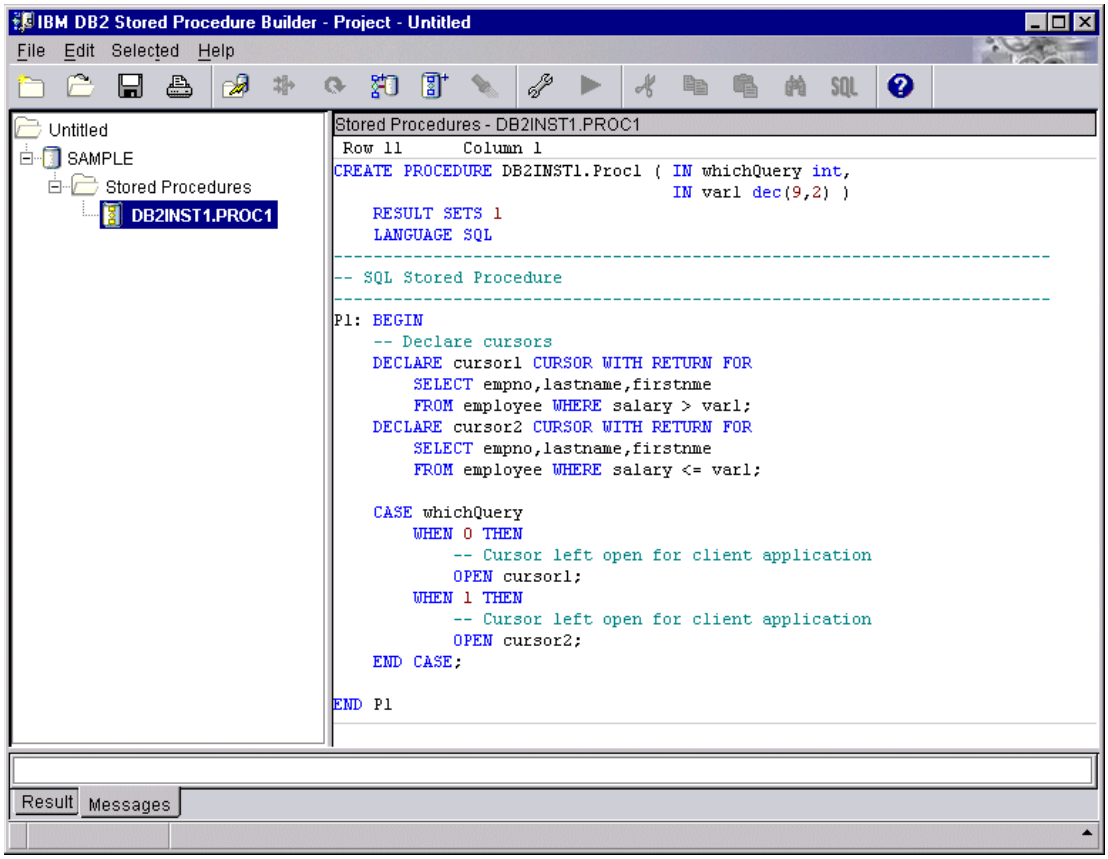

Figure 39. Stored Procedure Builder: generated procedure

The generated procedure code can be modified as you like before building the stored procedure. If you prefer either the vi or emacs editor, open the Environment Properties window from the File menu and set the **Editor** option.

To build the stored procedure and register to the database, click the **Build** icon from the icon bar.

If the building phase is completed successfully, you can test the stored procedure. Click the **Run** icon from the icon bar, and enter the input parameter values if necessary. You will see the test results as in the following example (Figure 40):

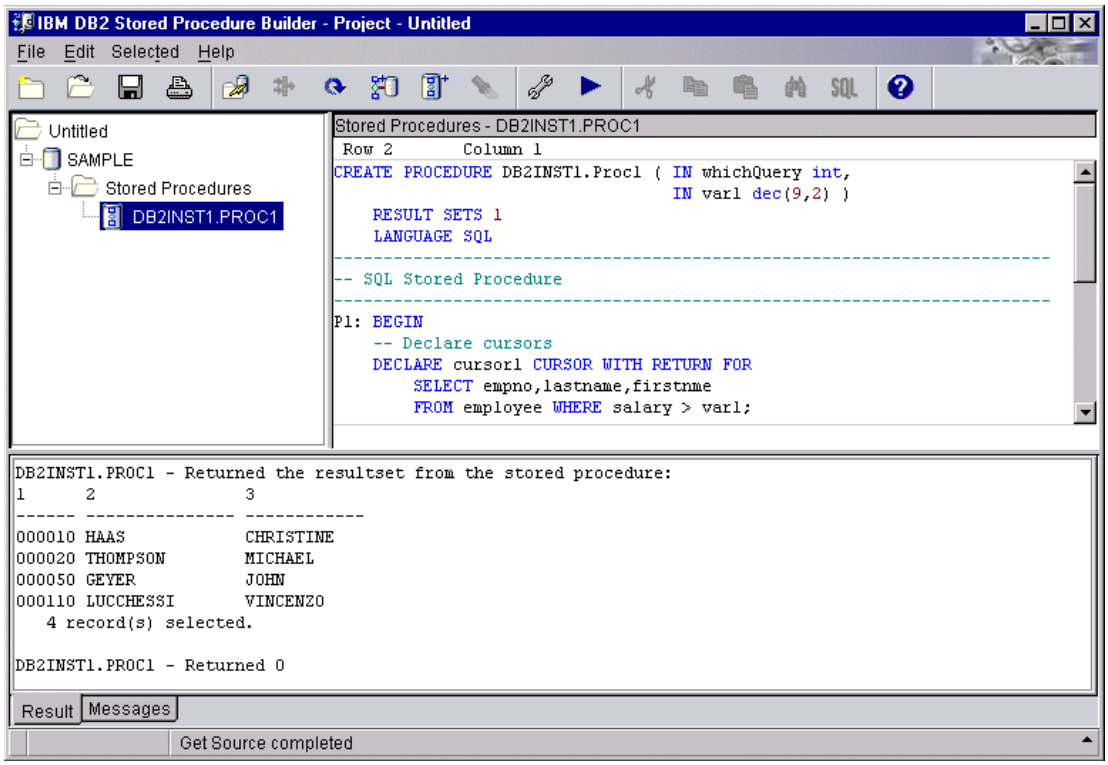

Figure 40. Stored Procedure Builder: testing a procedure

If you specify to use Java for the stored procedure in the selection shown in Figure 37 on page 195, the Java procedure using SQLJ or JDBC can be generated by SPB. Figure 41 shows a Java stored procedure using JDBC.

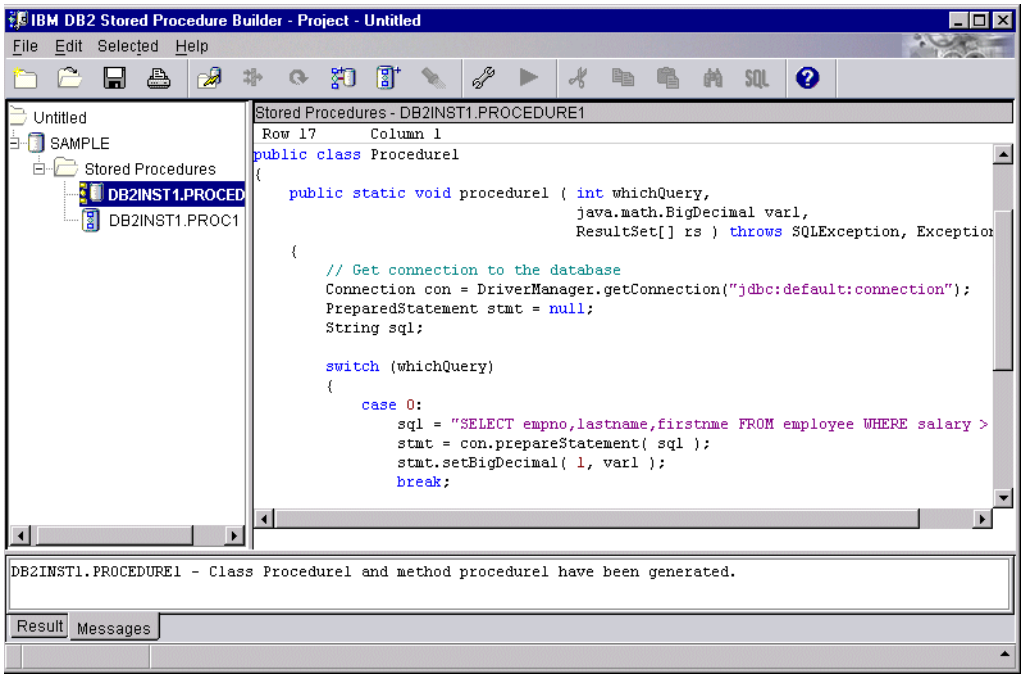

Figure 41. Stored Procedure Builder: generated Java procedure

7.9 Embedded SQL program conversion

This section discusses the conversion of embedded SQL programs from Sybase to DB2 UDB. These conversion topics are divided into the following sections:

- Statements comparison
- Connection
- Transaction
- Select statement
- Simple example
- Execute stored procedure
- SQLCA and SQLDA structure

7.9.1 Statements comparison

In both Sybase and DB2 UDB, almost of all the SQL statements are supported in embedded SQL programs. But some of Sybase's SQL statements are not supported in DB2 UDB. Table 28 shows the embedded SQL statement comparison between Sybase and DB2 UDB. Table 28 also shows which sections in this chapter you can refer to for more details.

Table 28. Embedded SQL statements comparison

Sybase	DB2 UDB	Related sections	Description
begin declare section	begin declare section		declare host variables used in a program
begin transaction	N/A	7.9.3	begin an explicit transaction
close	close		close a cursor
commit	commit	7.9.2 , 7.9.3	commit changes
connect	connect	7.9.2	connect to server (connect to database for DB2)
deallocate cursor	N/A		deallocate cursor
deallocate prepare	N/A		deallocate a prepared statement
declare cursor (dynamic/static)	declare cursor		declare a cursor with SQL statements
declare cursor (stored procedure)	N/A	7.9.6	declare cursor with a stored procedure
delete (positioned / searched)	delete		execute DELETE statement
describe input (SQLDA)	describe into	7.9.8	get information about parameter markers in a dynamic SQL statement
describe output (SQLDA)	describe into	7.9.8	get row format information about the result set of a dynamic SQL statement

Sybase	DB2 UDB	Related sections	Description
disconnect	disconnect	7.9.2	disconnect from server (disconnect from database for DB2)
exec	call		execute stored procedure
exec sql	exec sql		Marks the beginning of an SQL statement
execute	execute		executes a dynamic SQL statement from a prepared statement
execute immediate	execute immediate		executes a dynamic SQL statement
exit	N/A		close Client-Library and deallocates Embedded SQL resources
fetch	fetch		execute FETCH statement
get diagnostics	N/A		retrieves error or warning messages from Client-Library
include “filename”	include ‘filename’		include an external file
include sqlca	include sqlca	7.9.7	define SQLCA
include sqlda	include sqlda	7.9.8	define SQLDA
initialize_application	N/A		generate a call to set the application name
open (dynamic / static)	open		open a cursor
prepare	prepare		declare a name for a dynamic SQL statement
rollback	rollback	7.9.2 , 7.9.3	execute ROLLBACK statement
select	select	7.9.4	execute SELECT statement

Sybase	DB2 UDB	Related sections	Description
set connection	set connection		change the current connection
update	update		execute UPDATE statement
whenever	whenever		specify an action when a specific condition

SQL Descriptor

Sybase has two descriptors to store a description of the variables used in a prepared dynamic SQL statement. One is an SQL descriptor, and the other is an SQLDA. Since DB2 UDB has only SQLDA as a descriptor, if SQL descriptors are used in your Sybase programs, you need to convert the SQL descriptors to SQLDAs in DB2 UDB. The following statements are used for SQL descriptor in Sybase.

- allocate descriptor
- deallocate descriptor
- describe input (SQL descriptor)
- describe output (SQL descriptor)
- get descriptor
- set descriptor

7.9.2 Connection

Between Sybase and DB2 UDB, the meaning of a connection is different. In Sybase, a connection is established to a server, not to a database. After connecting to a server, the program can access the database objects with 3 parts name like '*database_name.owner_name.object_name*' or the `USE` *database* statement. In DB2 UDB, a connection is established to a database. The informations of servers and databases are defined in node directory and database directory in DB2 UDB. The syntax of the `CONNECT` statement is as follows:

In Sybase:

```
connect user_name [identified by password] [at connecction_name] [using
server_name]
```

In DB2 UDB:

```
connect to database_alias_name [user user_name] [using password]
```

In these statements, Sybase has a connection name as a parameter and DB2 UDB does not. Embedded SQL programs in both Sybase and DB2 UDB support multiple connections although the implementations are different.

In Sybase, you can specify a connection name for each SQL statement as follows:

```
exec sql [at connection_name] sql_statements
```

These SQL statements include the COMMIT and ROLLBACK statements, so you can issue the COMMIT or ROLLBACK statement for each connection. Sybase supports multiple connections to a single server or different servers.

In DB2 UDB, when your program connects to multiple databases, you need to use a type 2 connection in your program. The type 2 connection allows an embedded SQL program to connect to multiple databases.You can change the current connection to the others with SET CONNECTION statement, and the COMMIT and ROLLBACK statements affect all the connections which are established to the databases. If you need to execute COMMIT or ROLLBACK statements for each connection individually, you need to develop a multi-thread embedded SQL program or a CLI (Call Level Interface) program. For the detailed information, see the *Application Development Guide*, SC09-2949.

7.9.3 Transaction

Sybase has unchained mode and chained mode for transactions. In unchained mode, you need to issue the BEGIN TRANSACTION statements paired with the COMMIT TRANSACTION or ROLLBACK TRANSACTION statements explicitly to start and complete a transaction. In chained mode, Sybase starts a transaction implicitly before the following statements: SELECT, INSERT, UPDATE, DELETE, OPEN, and EXEC. The chained mode is the default mode in embedded SQL.

DB2 UDB has only a transaction mode which is equivalent to Sybase's chained mode.

In Sybase's chained mode, a transaction starts implicitly and you cannot issue the BEGIN TRANSACTION statement; however, you can change the transaction mode in your program and issue BEGIN TRANSACTION statement.

Some options of the COMMIT and ROLLBACK statements are different between Sybase and DB2 UDB. These options are as follows:

In Sybase:

```
COMMIT [transaction | tran | work] [transaction_name]
ROLLBACK [transaction | tran | work] [transaction_name | savepoint_name]
```

You cannot specify a transaction name in chained mode.

In DB2 UDB:

```
COMMIT [WORK]
ROLLBACK [WORK] [TO SAVEPOINT savepoint_name]
```

DB2 UDB does not have a transaction name as a parameter.

7.9.4 Select statement

The SELECT statements in Sybase and DB2 have almost the same functionality. However, in Sybase, the SELECT statement allows you to retrieve multiple rows with 1 SELECT statement as follows:

```
exec sql begin declare section;
    CS_CHAR titleid_array [100] [6];
exec sql end declare section;
...
exec sql select title_id into :titleid_array
    from titles;
```

In DB2 UDB, you cannot retrieve multiple rows with one SELECT statement in an embedded SQL program. When you want to retrieve multiple rows, you need to use DECLARE, OPEN, and FETCH statements with a cursor.

Note

You can manipulate multiple rows of data at a time if you use the DB2 Call Level Interface (CLI).

7.9.5 Example of embedded SQL program

We will show a simple embedded SQL program here. This example program executes the SELECT ... INTO statement that retrieves 1 row. When you convert this example to DB2, what you need to do is change the header file, the CONNECT statement, the USE statement, and the sqlca structure.

Here is the Sybase embedded SQL program:

```
#include <stdio.h>
#include "sybsqlex.h"   /* this header file is needed for only Sybase */
exec sql include sqlca;

main(int argc, unsigned char *argv[])
{
   exec sql begin declare section; /* definition of host variables */
        char username[30];
        char password[30];
        char servername[30];
   char au_id[12];
   char phone[13];
   short phone_ind;
   exec sql end declare section;

    if(argc>1){                    /* check command option -> au_id */
        printf("%s\n",argv[1]);
        strcpy(au_id,argv[1]);
    }else{return;}
    strcpy(username,"sa");        /* set username */
    strcpy(password,"pass");      /* set password */
    strcpy(servername,"ununbium");/* set servername */
    /* connect to server */
exec sql connect :username IDENTIFIED BY :password USING :servername;
if(check_error(sqlca.sqlcode)!=0){
        printf("Connection failed\n");
        return;
    }
    exec sql use pubs2;            /* select database for only Sybase */
    /* execute select statement */
    exec sql select phone into :phone:phone_ind from authors
      where au_id=:au_id;
    if(check_error(sqlca.sqlcode)!=0){
        printf("Select statement failed \n");
        exec sql disconnect all;
        return;
    }
    if(phone_ind==0){          /* check null indicator */
        printf("Phone No. : %d\n",phone);
    }else{
        printf("not exist\n");
    }
    exec sql disconnect;
}
```

```
check_error(int sqlcode)       /* check sqlcode */
{
    if(sqlcode!=0){
       fprintf(stderr,"\nSQLCODE:%5d \n** %s",
         sqlca.sqlcode,sqlca.sqlerrm.sqlerrmc);
        return(1);
    }else{
        return(0);
    }
}
```

Here is the converted embedded SQL program for DB2:

```
#include <stdio.h>
exec sql include sqlca;

main(int argc, unsigned char *argv[])
{
   exec sql begin declare section; /* definition of host variables */
        char username[30];
        char password[30];
        char servername[30];
   char au_id[12];
   char phone[13];
   short phone_ind;
   exec sql end declare section;

    if(argc>1){                    /* check command option -> au_id */
        printf("%s\n",argv[1]);
        strcpy(au_id,argv[1]);
    }else{return;}
    strcpy(username,"sa");         /* set username */
    strcpy(password,"");           /* set password */
    strcpy(servername,"ununbium"); /* set servername */
    /* connect to server */
   exec sql connect to :servername user :username using :password;
if(check_error(sqlca.sqlcode)!=0){
        printf("Connection failed\n");
        return;
    }
/* execute select statement */
    exec sql select phone into :phone:phone_ind from authors
      where au_id=:au_id;
    if(check_error(sqlca.sqlcode)!=0){
        printf("Select statement failed \n");
        exec sql disconnect all;
        return;
```

```
    }
    if(phone_ind==0){          /* check null indicator */
        printf("Phone No. : %d\n",phone);
    }else{
        printf("not exist\n");
    }
    exec sql disconnect all;
}

check_error(int sqlcode)       /* check sqlcode */
{
    if(sqlcode!=0){
       fprintf(stderr,"\nSQLCODE:%5d \n** %s",
         sqlca.sqlcode,sqlca.sqlerrmc);
        return(1);
    }else{
        return(0);
    }
}
```

Note that the CONNECT statement has been changed. In this example, the check_error function checks the sqlcode, and if the sqlcode is not 0, then prints the error message. When you want to get the messages from the sqlca, you need to use the variable sqlca.sqlerrm.sqlerrmc in Sybase, sqlca.sqlerrmc in DB2 UDB.

7.9.6 Executing a stored procedure

In both Sybase and DB2 UDB, embedded SQL programs can call stored procedures, but there are some differences between the methods used. The statement to call a stored procedure is as follows:

In Sybase:

```
exec [[status_var = ] status_value] procedure_name
[([[@param_name =] param_value [out[put]],...)]
[into :hostvar_1 [:indicator_1] [, hostvar_n [indicator_n,...]]]
```

or

```
exec sql declare cursor_name cursor for execute procedure_name
([[@param_name =]:host_var] [,[@param_name =]:host_var]...)
```

You can get the cursor that is defined in the called stored procedure with this statement.

In DB2 UDB:

```
call procedure_name | host_variable
[using descriptor descriptor_name] | ([host_variable,...])
```

In Sybase, you can use the `into` option to get data of 1 row from a stored procedure. In DB2 UDB, you need to use parameters specified as `OUT` to get data from a stored procedure.

Return code from stored procedure

In Sybase, you can get a return code from a stored procedure as follows:

```
exec sql :retstat = get_sum_discount :titleid,:total_discounts out;
```

In this example, `retstat` variable is set to return code from the procedure `get_sum_discount`.

In DB2 UDB, you can get the return code from stored procedures with '`sqlca.sqlerrd[0]`'. The following example shows how to get the return code:

```
exec sql get_sum_sidcount(:titldid,:total_discounts);
retstat = sqlca.sqlerrd[0];
```

Details of SQLCA are in "SQL Communication Area (SQLCA)" on page 210.

Result set from stored procedure

In Sybase, you can get an answer set from a stored procedure in an embedded SQL program. The following example shows handling an answer set with the `DECLARE CURSOR` statement:

```
exec sql begin declare section;
   CS_CHAR  b_titleid[7];
   CS_CHAR  b_tytle[65];
   CS_CHAR  b_type[13];
exec sql end decalre section;
exec sql include sqlca;
exec sql connect "sa";
exec sql use pubs2;
/* call stored procedure and declare cursor */
exec sql declare c1 cursor for execute prc_titles(:b_type);
exec sql open c1;
for(;;){
   exec sql fetch c1 into :b_titleid,:b_title;
   if(sqlca.sqlcode == 100) {break;}
   printf("    %8s %s\n",b_titleid,b_title);
}
exec sql close c1;
exec sql disconnect all;
```

Note that the `declare cursor` statement is executed to declare a cursor for the answer set from the stored procedure `prc_titles`.

Here we assume that this embedded SQL program call the following stored procedure:

```
create proc prc_titles(@b_type char(12))
as select title_id,title from titles where type=2b_type
```

DB2 UDB does not support returning answer sets from stored procedures in embedded SQL programs; however, you can use the CLI interface to get answer sets from a stored procedure in an embedded SQL program. The following example shows the use of CLI interface in embedded SQL program:

```
EXEC SQL INCLUDE SQLCA;          /* Declaration of SQLCA       */
EXEC SQL BEGIN DECLARE SECTION;  /* Host variables definition  */
   char b_titleid[7],b_title[65],b_type[13];
EXEC SQL END DECLARE SECTION;
SQLHANDLE henv,hdbc,hstmt;                          /* handles  */
SQLCHAR * stmt =(SQLCHAR *)"CALL prc_titles(?)";/* SQL statement */
SQLRETURN  clirc = SQL_SUCCESS;
SQLSMALLINT numCols;
EXEC SQL CONNECT to sample user db2inst1 using db2inst1;/*Connect*/
/* ============== start of CLI program ================== */
SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);          (1)
SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
SQLConnect(hdbc,NULL,SQL_NTS,NULL,SQL_NTS,NULL,SQL_NTS);       (2)
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt) ;
SQLPrepare(hstmt, stmt, SQL_NTS);                              (3)
strcpy(b_type,"trad_cook");
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,        (4)
   SQL_C_CHAR,0, 0, b_type, 0, NULL);
SQLExecute(hstmt);                                             (5)
SQLNumResultCols(hstmt, &numCols) ;                            (6)
SQLBindCol(hstmt, 1, SQL_C_CHAR, b_titleid, 6, NULL);          (7)
SQLBindCol(hstmt, 2, SQL_C_CHAR, b_title, 64, NULL);
clirc = SQLFetch(hstmt);                                       (8)
printf("\n--ID--,----Title---- \n");
while(clirc != SQL_NO_DATA_FOUND)
{  printf("%6s,%64s\n", b_titleid, b_title);
   clirc = SQLFetch(hstmt);
}
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);                         (9)
/* ================ end of CLI program ================== */
EXEC SQL commit;
EXEC SQL connect reset;       /* Disconnect */
```

1. Allocating handles of CLI for environment and connection.
2. Connecting to server (database) with NULL value. After this statement, CLI statements can use the connection which was established by the embedded SQL's `CONNECT` statement.
3. Preparing the SQL statement that calls the stored procedure '`prc_titles`'. Binding the SQL statement to a database server.
4. Binding the '`b_type`' variable for the parameter marker '?' in the SQL statement.
5. Executing the SQL statement.
6. Checking the number of columns in the result set.
7. Binding variables for the answer set.
8. Fetching each row of the answer set.
9. Deallocating statement handle.

7.9.7 SQL Communication Area (SQLCA)

Both Sybase and DB2 UDB have the SQL Communication Area (SQLCA) structure to store the return code for each SQL statement. As shown in Table 29 and Table 30, the names and functionality are almost same, with some differences in the parameter types and the usages. For example, `sqlcode` is an integer type value in Sybase, while it is a short value in DB2 UDB. The field `sqlerrd[2]` in Sybase shows the number of rows affected whereas DB2 does not have this information in SQLCA. The `sqlerrd[0]` in DB2 UDB has the return code from a called stored procedure.

Table 29. Fields of SQLCA structure in Sybase

Field	Data type	Descriptors
sqlcaid	char	ID number of SQLCA
sqlcabc	integer	Size of SQLCA in bytes
sqlcode	integer	SQL return code
sqlerrm.sqlerrml	integer	Length for SQLERRMC
sqlerrm.sqlerrmc	char	Error message tokens
sqlerrp	char	Diagnostic information
sqlerrd	integer[6]	Diagnostic information [2] is the number of rows affected.
sqlwarn	char	Warning flags

Table 30. Fields of SQLCA structure in DB2 UDB

Field	Data type	Descriptors
sqlcaid	char	ID number of SQLCA
sqlcabc	integer	Size of SQLCA in bytes
sqlcode	short	SQL return code
sqlerrml	short	Length for SQLERRMC
sqlerrmc	char	Error message tokens
sqlerrp	char	Diagnostic information
sqlerrd	integer[6]	Diagnostic information
sqlwarn	char	Warning flags
sqlstate	char	Sqlstate

7.9.8 SQL Descriptor Area (SQLDA)

Both Sybase and DB2 UDB have the SQL Descriptor Area (SQLDA) to store the description of the variables used in a prepared dynamic SQL statement.

As shown in Table 31 and Table 32, the names of the structure members are different between Sybase and DB2. The sqlvar[].field in DB2 UDB and the `sd_column[].sd_datafmt` field in Sybase are used to get the data type of the column.

Table 31. Fields of SQLDA structure in Sybase

Field	Data type	Description
sd_sqln	short	Size of the sd_column array
sd_sqld	short	The number of columns in the query being described
sd_column[].sd_datafmt	CS_DATAFMT	Identifies the Client-Library structure associated with column
sd_column[].sd-sqldata	pointer	Address of host variable
sd_column[].sd_sqlind	short	NULL indicator for column
sd_column[].sd_sqllen	integer	Actual size of the data pointed by sd_sqldata
sd_column[].sd_sqlmore	pointer	Reserved by Sybase

Table 32. Fields of SQLDA structure in DB2 UDB

Field	Data type	Descriptors
sqldaid	char	ID number of SQLDA
sqldabc	integer	Size of SQLDA in bytes
sqln	short	The number of sqlvar elements
sqld	short	The number of columns or host variable
sqlvar[].sqltype	short	Data type of column
sqlvar[].sqllen	short	Data length of column
sqlvar[].sqldata	pointer	Pointer to variable data value
sqlvar[].sqlind	pointer	Pointer to Null indicator

7.10 Client-Library program conversion

Sybase Open Client product provides DB-Library and Client-Library, which includes a set of functions or APIs callable from third generation programming languages such as C. Using either DB-Library or Client-Library, you can write client application programs which send queries to Sybase server and process the results. Sybase encourages customers to use Client-Library rather than DB-Library for new application development, because DB-Library is an older interface. Client-Library was introduced with Sybase System 10.

DB2 UDB provides the environment to develop callable SQL applications using the DB2 UDB Call Level Interface (CLI). This driver implements the ODBC function set, with the exception of some extended functions implemented by the Driver Manager. If you have client application programs using DB-Library or Client-Library, you would convert them into DB2 CLI applications.

Coding DB2 CLI applications involves writing C/C++ modules that contain CLI functions. To complete the client application conversion successfully, you need to be familiar with the purpose, syntax, arguments and usages of CLI functions, as well as those of DB-Library or Client-Library functions.

In this section, we will be focusing on Client-Library, and examine Client-Library and DB2 CLI application development using a simple example which fetches rows from a table and display them.

7.10.1 Initialization and termination

A Client-Library application and a DB2 CLI application can be broken down into a set of tasks. The basic tasks are: Initialization, SQL statement processing, and Termination.

7.10.1.1 Initialize environment for a Client-Library application

In a Client-Library application, the initialization tasks involves allocating and initializing environment, and establishing a connection to the server.

As the first step, a programming context structure needs to be allocated. To request a context structure, an application calls the `cs_ctx_alloc` function.

The next step is initializing Client-Library for the allocated programming context. The application calls the `ct_init` function which sets up internal control structures and defines the version of Client-Library.

Once the Client-Library has been initialized, you can establish a connection to a server by the following steps:

- Allocate a connection structure

  A connection structure should be allocated by the `ct_con_alloc` function.

- Set properties for the connection (such as user name and password) if necessary

  To set user name and password for the allocated connection structure, you should execute the `ct_con_props` function with an action type of `CS_SET`, for the `CS_USERNAME` property and the `CS_PASSWORD` property.

- Logs into a server

  Execute the `ct_connection` function to establish a connection to a server. The argument to this function includes the allocated connection structure and the server name.

7.10.1.2 Terminate environment for a Client-Library application

The termination phase involves disconnecting your application from the server and freeing allocated resources after the transaction processing has completed. The `ct_close` function closes a connection. The corresponding connection structure can then be freed using `ct_con_drop` function. Once all the connection structures have been freed, the `ct_exit` function can be called to free the resource allocated for the Client-Library. If there are open connections, the `ct_exit` function will close and deallocate all the connections. Finally the Client-Library application calls the `cs_ctx_drop` function to deallocate the context structure which allocated at the beginning of the program.

Figure 42 shows the function call sequences for initialization and termination.

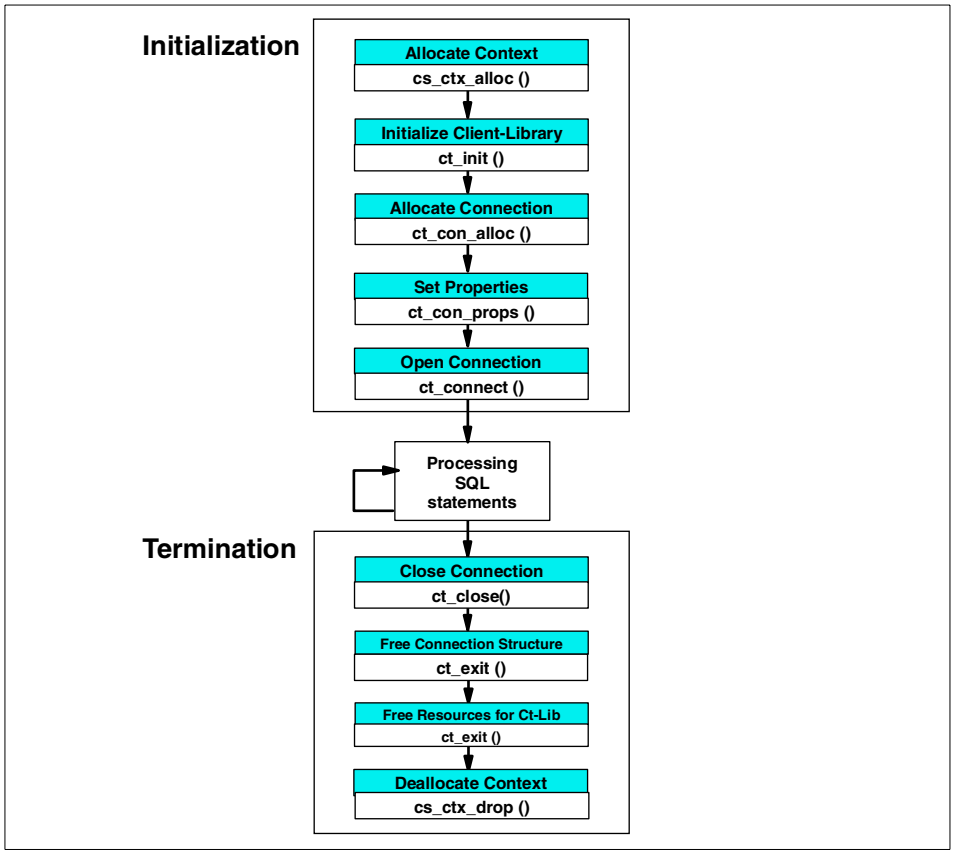

Figure 42. Client-Library initialization and termination

Here is a C code sample illustrating the initialization and termination phase:

```
int main( ) {
   CS_CONTEXT *context;
   CS_CONNECTION *connection;

   /* allocate a context structure*/
   cs_ctx_alloc(CS_VERSION100, &context);
   /* initialize Client-Library */
   ct_init( context, CS_VERSION100);
   /* allocate a connection structure */
   ct_con_alloc ( context, &connection );
   /* setting user name and password for opening the connection*/
   ct_con_props ( connection, CS_SET, CS_USERNAME,
                  "sa",CS_NULLTERM, NULL);
```

```
    ct_con_props ( connection, CS_SET, CS_PASSWORD,
                   "pswd",CS_NULLTERM, NULL);
    /* connect to the data source */
     ct_connect ( connection,"ununbium",0);

    /*********   Start SQL statements processing  ***********/
    /* allocate command structure, execute statement, etc.*/
    /*********   End SQL statements Processing **************/

    /* disconnect from server*/
    ct_close (connection, CS_UNUSED);
    /* free the connection structure */
    ct_con_drop ( connection );
    /* free the resource for Client-Library*/
     ct_exit( context, CS_UNUSED ) ;
    /* deallocate context */
    cs_ctx_drop ( context ) ;

    exit ( 0 ) ;
}
```

7.10.1.3 Initialize environment for a CLI application

Like Client-Library applications, a CLI application needs to perform an initialization task first. As the first step for an application that uses DB2 CLI, an application needs to allocate an environment handle. An environment handle provides access to global information such as attributes and connections.To allocate an environment handle, the `SQLAllocHandle` function should be called with a handle type of `SQL_HANDLE_ENV`. DB2 UDB CLI allocates the environment handle, and passes the value of the associated handle back.

The next step is allocating a connection handle. A connection handle refers to a data object that contains information associated with a connection to a particular data source. This includes connection attributes, general status information, transaction status, and diagnostic information.To request a connection handle, an application calls `SQLAllocHandle` with a handle type of `SQL_HANDLE_DBC`. You may need to allocate several connections handles in your application in order to connect to more than one database, or even to establish multiple connections to the same database. That is, one connection handle for each concurrent connection.

Once a connection handle has been allocated, you can attempt to establish a connection to the data source using that connection handle. The function `SQLConnect` is used to request a database connection. The arguments to this function include the name of the target data source, and optionally a useID and password.

7.10.1.4 Terminate environment for a CLI application

The termination phase involves disconnecting your application from the database(s) and freeing allocated resources after the transaction processing has completed. The `SQLDisconnect` API closes a connection. The corresponding connection handle can then be freed using `SQLFreeHandle` with a handle type of `SQL_HANDLE_DBC`. Only once all the connections handles have been freed, the `SQLFreeHandle` function can be called with a handle type of `SQL_HANDLE_ENV` to successfully free the environment handle.

Figure 43 shows the function call sequences for initialization and termination.

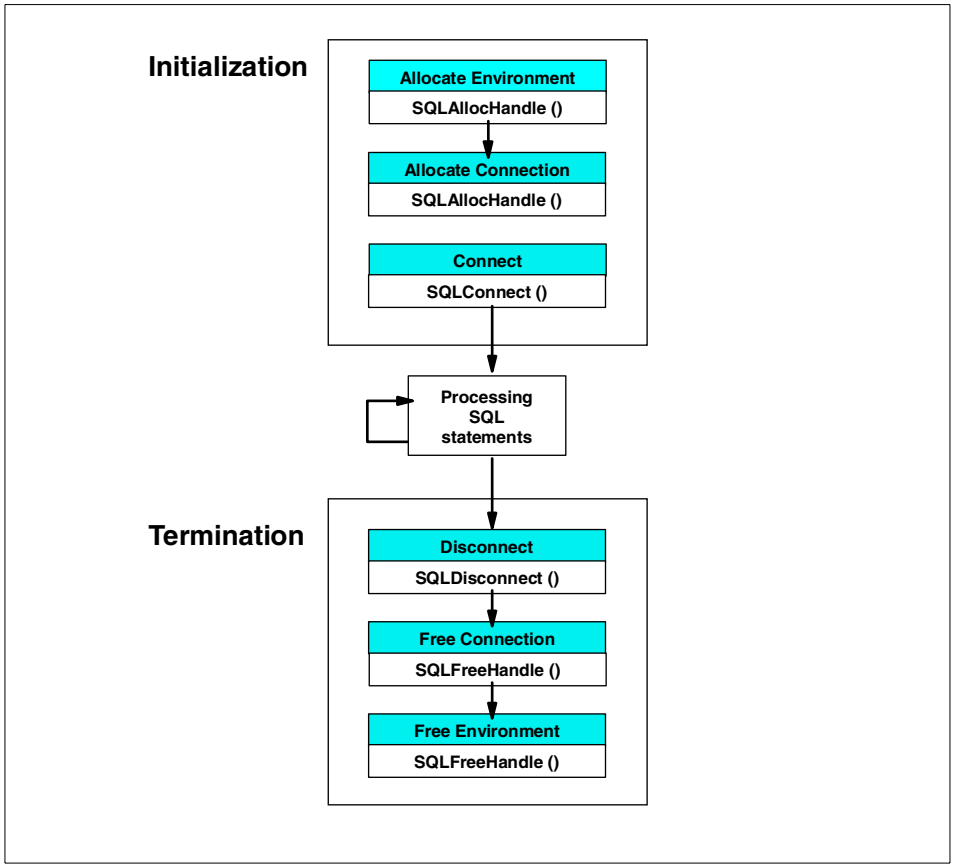

Figure 43. CLI initialization and termination

Here is a C code sample illustrating the initialization and termination phase:

```
int main( ) {
    SQLHANDLE henv;
    SQLHANDLE hdbc;

    /* allocate an environment handle */
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
    /* allocate the connection handle */
    SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc );
    /* connect to the data source */
    SQLConnect( hdbc, "pubs2", SQL_NTS, "userid", SQL_NTS,
                "password", SQL_NTS );

    /*********   Start SQL statements processing  ***********/
    /* allocate statement handle, execute statement, etc.*/
    /*********   End SQL statements Processing **************/

    /* disconnect from database */
    SQLDisconnect( hdbc ) ;
    /* free the connection handle */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc ) ;
    /* free environment handle */
    SQLFreeHandle( SQL_HANDLE_ENV, henv ) ;
     return ( SQL_SUCCESS ) ;
 }
```

7.10.2 Executing SQL statement

The main task of the application is accomplished during the SQL statement processing phase. Once a connection has been established to a data source, the application can submit SQL statements to the data source.

7.10.2.1 Execute SQL statements for a Client-Library application

SQL statements are passed to Client-Library to query and retrieve the data using a four step process:

1. Allocating command structure(s)
2. Building and executing SQL statements
3. Processing results
4. Deallocating the command structure

Allocating command structure(s)

A Client-Library application allocates command structures before executing SQL statements. A Client-Library application uses command structures to

send SQL statements to a server and process the results. A command structure can be allocated executing the ct_cmd_alloc function.

Building and executing SQL statements
Once a command structure has been allocated, SQL statement can be specified and executed. An application uses the ct_command function with a command type of SQL_LANG_CMD to build the SQL statement to be submitted. This function associates a specified SQL statement with the allocated command structure.

To submit the built SQL statement, the ct_send function is used.

Processing results
After the statement has been executed, an application calls the ct_result function to deal with the result. Executing this function, you can get the type of result data. Depending on the type of result data, you need to perform different processing.

For example, if you execute a select statement, the ct_result function would get a result type of CS_ROW_RESULT, which indicates that zero or more rows of a result set has been returned. Then the number of columns in the result set can be found using the ct_res_info function. Information about the columns in the result set, like name, column type or length can be obtained using ct_describe.

In order to receive the data into variables in the application, the application executes the ct_bind function. This function binds the application variables to columns in the result set.

The next step is to call the ct_fetch function to fetch the first or next row of the result set. If any columns have been bound in the previous step, data is received into the bound application variables.

Deallocating the command structure
The ct_cmd_drop function is used to end processing for that SQL statement. This drops the command structure.

Figure 44 shows the function sequences for SQL statement processing.

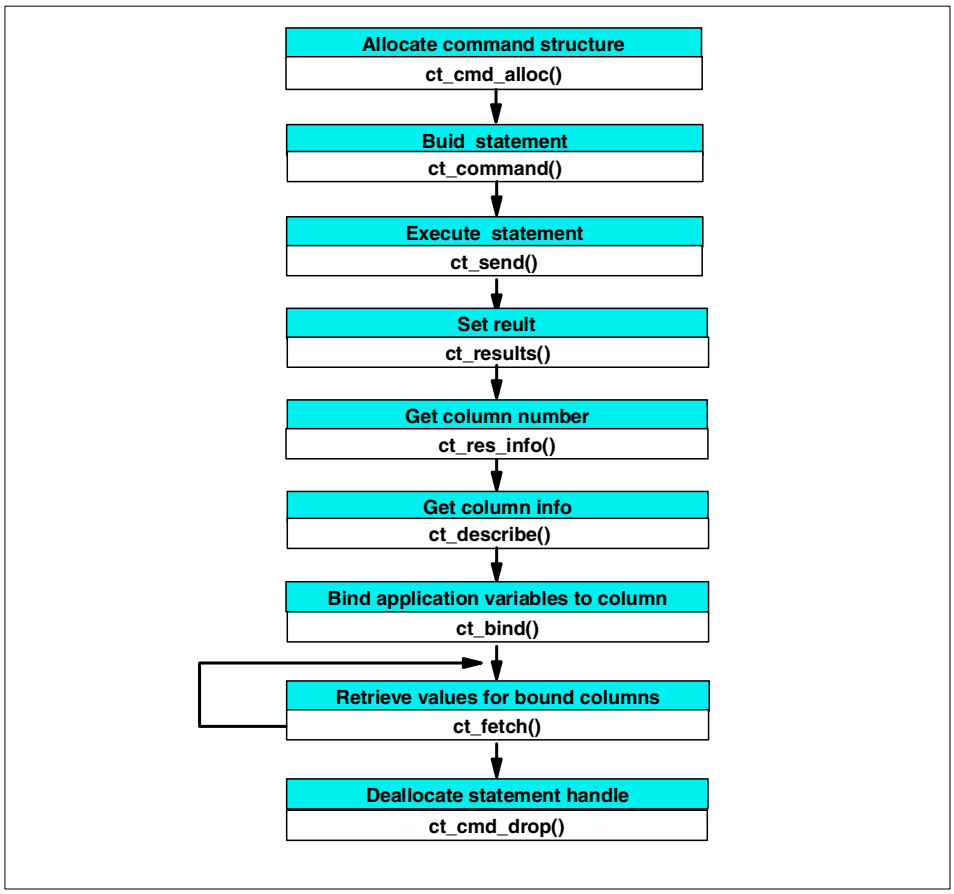

Figure 44. Select statement processing in a Client-Library application

Here is a C code sample illustrating the SQL processing phase:

```
int main( ) {
   CS_CONTEXT     *context; /* context structure */
   CS_CONNECTION  *connection; /* connection structure */
   CS_COMMAND     *cmd; /* command structure */

   /* Data format structures for column descriptions: */
   CS_DATAFMT     columns [2];

   CS_INT         datalength[2];
   CS_SMALLINT    indicator[2];
   CS_INT         count;
```

```
   CS_INT         results_type;
   CS_CHAR        name[40];
   CS_CHAR        city[40];

   /******************** Initialization ****************/
   /* allocate contest, connection structure, connect to the server */
   /***************** End of Initialization **************/

/*********   Start SQL statement Processing  ************/
   /* allocate command structure */
   ct_cmd_alloc(connection, &cmd);
   /* build SQL statement */
   ct_command(cmd, CS_LANG_CMD,
         "select au_lname, city from pubs2..authors where state = 'CA'",
          CS_NULLTERM, CS_UNUSED);
   /* exeucte the statement */
   ct_send(cmd);
   /* set up the result data */
   ct_results(cmd,&result_type);
   /* bind the first (au_lname) column of result set */
   columns[0].datatype = CS_CHAR_TYPE;
   columns[0].format = CS_FMT_NULLTERM;
   columns[0].maxlength = 40;
   columns[0].count = 1;
   columns[0].locale = NULL;
   ct_bind(cmd, 1, &columns[0], name, &datalength[0],&indicator[0]);
   /* bind the second (city) column of result set */
   columns[1].datatype = CS_CHAR_TYPE;
   columns[1].format = CS_FMT_NULLTERM;
   columns[1].maxlength = 40;
   columns[1].count = 1;
   columns[1].locale = NULL;
   ct_bind(cmd, 2, &columns[1], city, &datalength[1], &indicator[1]);
   /* fetch and print each row */
   while ( ct_fetch(cmd, CS_UNUSED, CS_UNUSED, CS_UNUSED, &count)
         == CS_SUCCEED)
   printf("%s : %s \n", name, city );

   /* deallocate the command structure */
   ct_cmd_drop(cmd);
   /*********   End statement processing **************/

   /******************** Termination ********************/
   /* disconnect, free connection and context structure*/
   /*****************************************************/
exit (0);
}
```

7.10.2.2 Execute SQL statements for a CLI application

SQL statements are passed to CLI to query and retrieve the data using a four step process:

1. Allocating statement handle(s)
2. Preparing and executing SQL statements
3. Processing results
4. Freeing statement handle(s)

Allocating statement handle(s)

Statement handles need to be allocated before any SQL statements can be executed. A statement handle refers to the data object that is used to track the execution of a single SQL statement. This includes information such as statement attributes, SQL statement text, cursor information, result values and status information. The API `SQLAllocHandle` is called with a handle type of `SQL_HANDLE_STMT` to allocate a statement handle.

Preparing and executing SQL statements

Once a statement handle has been allocated, SQL statements can be specified and executed using one of two methods:

- Execute directly, which combines the prepare and execute steps into one. You would use this method if the statement will be executed only once or if the column information is not needed prior to statement execution. The function `SQLExecDirect` is used for executing the statement directly.
- Prepare then execute, which splits the preparation of the statement from the execution. This method is useful if the statement will be executed repeatedly, usually with different parameter values. This avoids having to prepare the same statement more than once. The subsequent executions make use of the access plans already generated by the prepare. The prepare followed by execute is accomplished by the function calls `SQLPrepare` and `SQLExecute`.

Processing results

The next step after the statement has been executed depends on the type of SQL statement. If the statement is a query, you usually need to perform the following steps to retrieve each row of the result set:

The first step requires establishing or describing the structure of the result set. The number of columns in the result set is found using `SQLNumResultCols`. Information about the columns in the result set, like name, column type or length is obtained using `SQLDescribeCol` or `SQLColAttributes`.

In order to receive the data into the application, one option is to bind the application variables to columns in the result set using `SQLBindCol`.

The third step is to call `SQLFetch` to fetch the first or next row of the result set. If any columns have been bound in the second step, data is received into the bound application variables.

If the application does not bind any columns in the second step, as in the case when it needs to retrieve columns of long data in pieces, it can use `SQLGetData` after the fetch.

Both the `SQLBindCol` and `SQLGetData` techniques can be combined if some columns are bound and some are unbound.

Freeing statement handle(s)

The call `SQLFreeHandle` with a handle type of `SQL_HANDLE_STMT` is used to end processing for that statement. This drops the statement handle, and releases all associated resources. However it is not necessary to drop the statement handle if the associated SQL statement needs to be executed again by the application.

Figure 45 shows the function sequences for SQL statement processing.

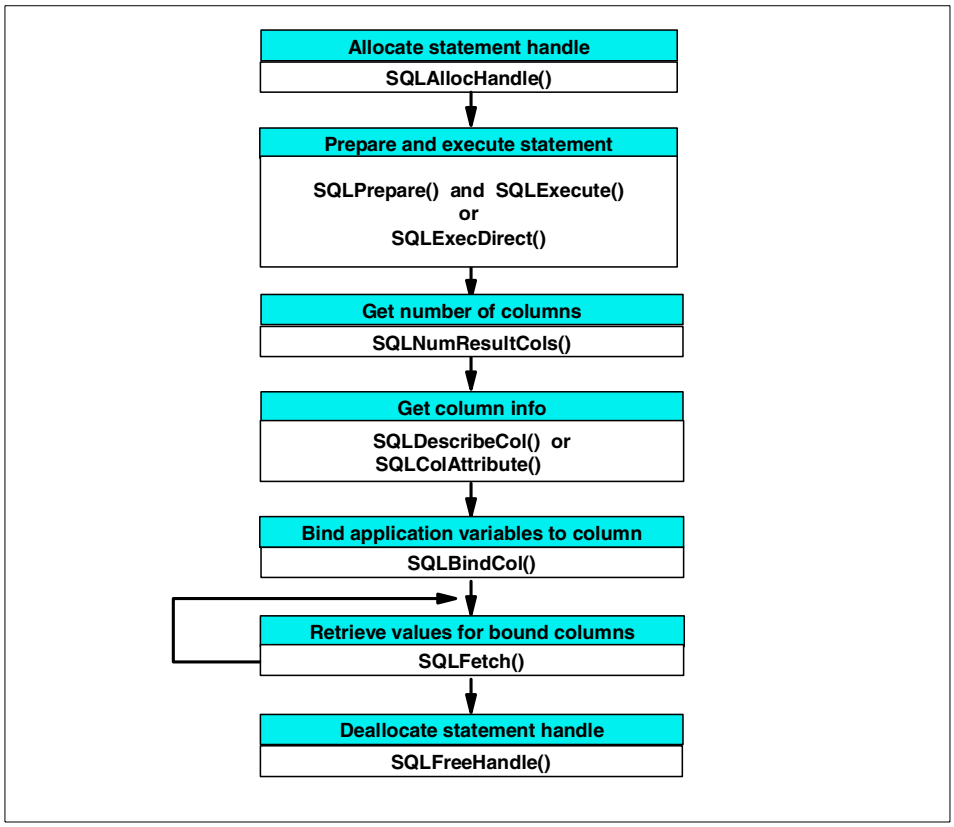

Figure 45. Select statement processing in a CLI application

Here is a C code sample illustrating the SQL processing phase:

```
int main( ) {
    SQLHANDLE henv; /*environemnt handle */
    SQLHANDLE hdbc; /* connection handle */
    SQLHANDLE hstmt; /* statement handle */

   SQLCHAR * stmt = ( SQLCHAR * ) "SELECT au_lname, city
                       FROM authors WHERE state = 'CA'";
   struct
   {  SQLINTEGER ind ;
      SQLCHAR val[41];
   } au_lname; /* variable to be bound to the AU_LNAME column */

   struct
```

```
   {  SQLINTEGER ind ;
      SQLCHAR val[21];
   } city; /* variable to be bound to the CITY column */

    /******************** Initialization ****************/
    /* allocate henv, hdbc, connect to the database       */
    /***************** End of Initialization **************/

    /*********   Start SQL statement Processing  ************/
   /* set AUTOCOMMIT on */
   SQLSetConnectAttr (hdbc, SQL_ATTR_AUTOCOMMIT,
                      (SQLPOINTER) SQL_AUTOCOMMIT_ON, SQL_NTS) ;
   /* allocate statement handle */
   SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt );
   /* exeucte the statement */
   SQLExecDirect(hstmt, stmt, SQL_NTS);
   /* alternatively, we could have prepared and executed the statement*/
   /* SQLPrepare (hstmt, stmt, SQL_NTS);  */
   /* SQLExecute (hstmt); */
   /* for the sake of simplicity, we will leave out steps */
   /*             to find out the structure of result set */
   /* bind the first (au_lname) column of result set */
   SQLBindCol(hstmt, 1, SQL_C_CHAR, au_lname.val, 40, &au_lname.ind);
   /* bind the second (city) column of result set */
   SQLBindCol(hstmt, 2, SQL_C_CHAR, city.val, 20, &city.ind);

    /* fetch and print each row */
    while ( SQLFetch(hstmt) == SQL_SUCCESS )
         printf("%s : %s \n", au_lname.val, city.val );
   /* if you do not use the auto-commit mode, the transaction should */
   /* be committed explicitly as following: */
   /* SQLEndTran (SQL_HANDLE_DBC,hdbc, SQL_COMMIT); */

   /* free the statement handle */
   SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
   /*********   End statement processing **************/

   /******************** Termination ********************/
   /* disconnect, free connection and environment handles*/
   /*****************************************************/
return(SQL_SUCCESS);
}
```

7.10.3 Diagnostics and processing errors in CLI programs

In a CLI program, the error and warning handling should be done using function return codes and detailed diagnostics. Every CLI function returns a function return code. This provides basic diagnostic information to the application. The application should examine the return code before proceeding to the next function call. Table 33 lists possible CLI return codes. Not all return codes are applicable to each CLI function. If the application encounters an unexpected return code from a function call, typically anything other than `SQL_SUCCESS`, it can call `SQLGetDiagRec` or `SQLGetDiagField` to retrieve detailed diagnostic information. The details include SQLSTATE, the native error or SQLCODE, and the message text.

Table 33. DB2 UDB CLI function return codes

Return Code	Explanation
SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.
SQL_SUCCESS_WITH_INFO	The function completed successfully, with a warning or other information. Call `SQLGetDiagRec` to receive the SQLSTATE and any other informational messages or warnings.
SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed. The DB2 CLI driver has returned control to the application after calling the function, but the function has not yet finished executing.
SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling `SQLGetDiagRec`.
SQL_NEED_DATA	The application tried to execute an SQL statement but DB2 CLI lacks parameter data that the application had indicated would be passed at execute time.
SQL_ERROR	The function failed. Call `SQLGetDiagRec` to receive the SQLSTATE and any other error information.
SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.

Embedded SQL applications rely on the SQLCA for all diagnostic information. CLI applications can retrieve much of the same information by using `SQLGetDiagRec` hence it is not necessary to examine the SQLCA in most cases. However if you need to examine the SQLCA from within a CLI application the `SQLGetSQLCA` function can be used.

The following code illustrates error handling and retrieval of diagnostic information by a CLI application:

```
/* call a CLI function, for example: */
rc = SQLConnect( hdbc, "pubs2", SQL_NTS,
                 "baduid", SQL_NTS,"badpwd", SQL_NTS );
/* if not success, call an error checking routine */
if ( rc !=SQL_SUCCESS ) errprint(SQL_HANDLE_DBC, hdbc, rc);
...
/* example of a simple error routine to print the error: */
SQLRETURN errprint( SQLSMALLINT htype, /* A handle type */
                    SQLHANDLE   hndl,  /* A handle */
                    SQLRETURN   erc)   /* Return code */
{
    SQLCHAR     buffer[SQL_MAX_MESSAGE_LENGTH + 1] ;
    SQLCHAR     sqlstate[SQL_SQLSTATE_SIZE + 1] ;
    SQLINTEGER  sqlcode ;
    SQLSMALLINT length, i=1 ;

    printf( ">--- ERROR -- RC = %d ------------\n", erc);
    while ( SQLGetDiagRec( htype,hndl,i,sqlstate,&sqlcode,
                           buffer,SQL_MAX_MESSAGE_LENGTH + 1,
                           &length) == SQL_SUCCESS ) {
       printf( "         SQLSTATE: %s\n", sqlstate ) ;
       printf( "Native Error Code: %ld\n", sqlcode ) ;
       printf( "%s \n", buffer ) ;
       i++ ;
    }

    return( SQL_ERROR ) ;
}
```

In this example, we check to see whether the CLI call to connect to the database returned successfully. If the wrong userid/password is specified, an error condition occurs, and the `errprint` function is called to print out the diagnostic information.

7.10.4 Setup of environment for CLI application programs

When you develop CLI application programs, make sure you have installed the DB2 UDB Software Developer's Kit and bound the CLI bind files to the database. To bind the CLI bind files, perform the following command from the command line after connecting to the database:

```
db2 bind @db2cli.lst messages db2cli.msg grant public
```

In this example, `db2cli.lst` is the list file containing the CLI bind files. You can find the `db2cli.lst` file in the `bnd` directly under the product installed directory (default is `C:\Program Files\sqllib\bnd`) in Windows or OS/2 environments. In UNIX environments, the file is in the `$HOME/sqllib/bnd` directly where `$HOME` is the instance owner's home directly.

To change the default CLI behavior or increase CLI application performance, you may want to modify the DB2 CLI configuration file (`db2cli.ini`), in which you can specify the CLI configuration keywords. The DB2 CLI configuration file is located in the `cfg` directory under the product installed directory (Windows or OS/2 environment), or in the `$HOME/sqllib/cfg` directory where `$HOME` is the instance owner's home directly (UNIX environment).

You can directly edit the configuration file to set the CLI configuration keywords, or use the Client Configuration Assistant (CCA) on Windows or OS/2 platforms.

Appendix A. Conversion tools

In this appendix we will cover some of the conversion tools available to aid the conversion process. This is not intended to be a complete list, but covers some of the major products available to ease the conversion effort.

A.1 SQL Conversion Workbench

SQL Conversion Workbench (SQL-CW) Version 3.0, which is also called the Stored Procedure Conversion Tool (SProCT), is a Windows based conversion tool from ManTech Systems Solutions Corporation. It facilitates the conversion of various database and the associated applications to any database in the DB2 UDB family on any supported platform. Sybase Adaptive Server Version 12.0 is one of the database management systems supported. SProCT can be downloaded from the following Web site:

```
http://www-4.ibm.com/software/data/db2/migration
```

SProCT does the following:

- Unloads the metadata from the source database
- Loads the metadata into the repository database
- Edits the metadata
- Generates unload scripts for the source database
- Generates DDLs and load scripts for the target database
- Does data migration
- Does stored procedure conversion
- Provides metrics for the source database in order to estimate cost and effort
- Allows modifications to the design and data definition of the target database
- Allows code re-engineering
- Generates procedural code for stored procedures

See *SQL Conversion Workbench User Guide*, Document Number: 033100SProCT30, for more information on how to use SProCT. Once you have installed SProCT, you can find the PDF file of this manual in the following directory on your workstation:

```
C:\Program Files\mssc\scw\version3.pdf
```

This appendix will not cover all options and functionality of the tool, but will present a high level view of the process we followed for our conversion.

A.1.1 Installation overview

The installation of the SQL-CW product is well documented so we will not cover this in detail. It is either loaded from CD or from the file downloaded from the Web site mentioned previously. From the Windows start menu, select **RUN**, then the location of the file or CD and execute `\SETUP`, for example `F:\SETUP` and the installation screen will appear. Simply follow the instructions to complete the installation.

Note that a prerequisite to installation is that DB2 UDB Personal Edition, Workgroup Edition or Enterprise Edition be installed since SQL-CW uses DB2 for the repository database.

When you select **Start > Programs >SQL Conversion Workbench** from your Windows start menu, you will see six options:

- First Steps
- Meta Load
- Oracle Unload
- Repository Edit
- SQL Server Unload
- Workbench

To get started, select **First Steps** and you will a the screen as shown in Figure 46.

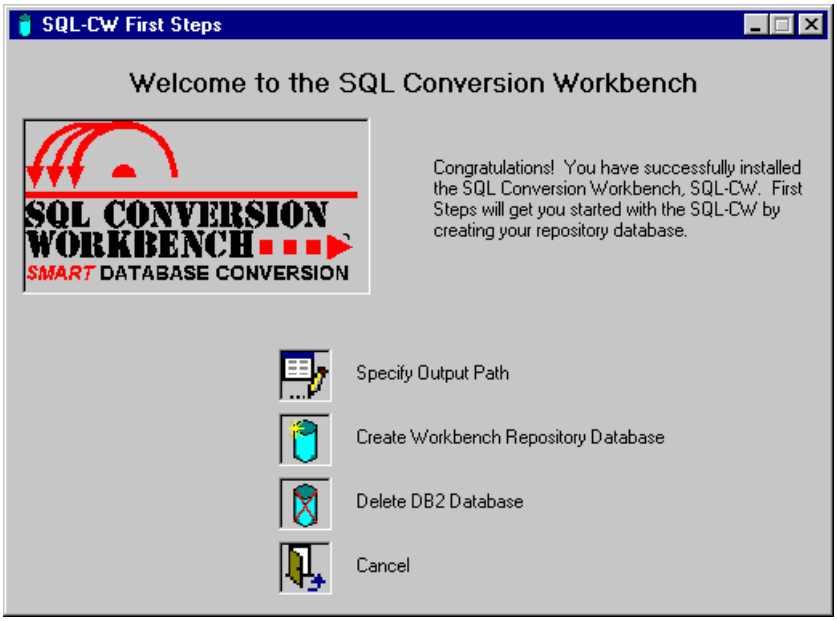

Figure 46. SQL-CW First Steps menu

If you have not already done so during the installation process, here you can specify the path for your repository database, create a repository database, or delete a DB2 database created previously.

If you have not already done these steps, they need to be done now.

When the repository database is completed, you can unload your metadata using either the **SQL Server Unload** option from the menu, or you can select the **SQL Conversion Workbench** from the main screen tool-bar.

A.1.2 Unloading metadata

The next major will be to unload the metadata. The source database must be registered as an ODBC data source before you can unload the metadata. To unload the metadata select **START > Programs > SQL Conversion Workbench > SQL Server Unload**, or click the **SQL Server Metadata Unload** icon on the main Window. This will bring up a window for you to login to your source database (Figure 47).

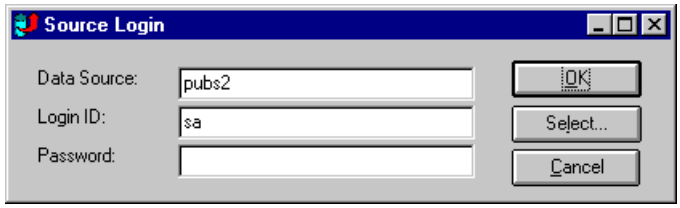

Figure 47. Login window for SQL Server unload

Change any incorrect information and click **OK**. You will next see a window of objects to select for the unload. You will be asked to specify output file names also (Figure 48).

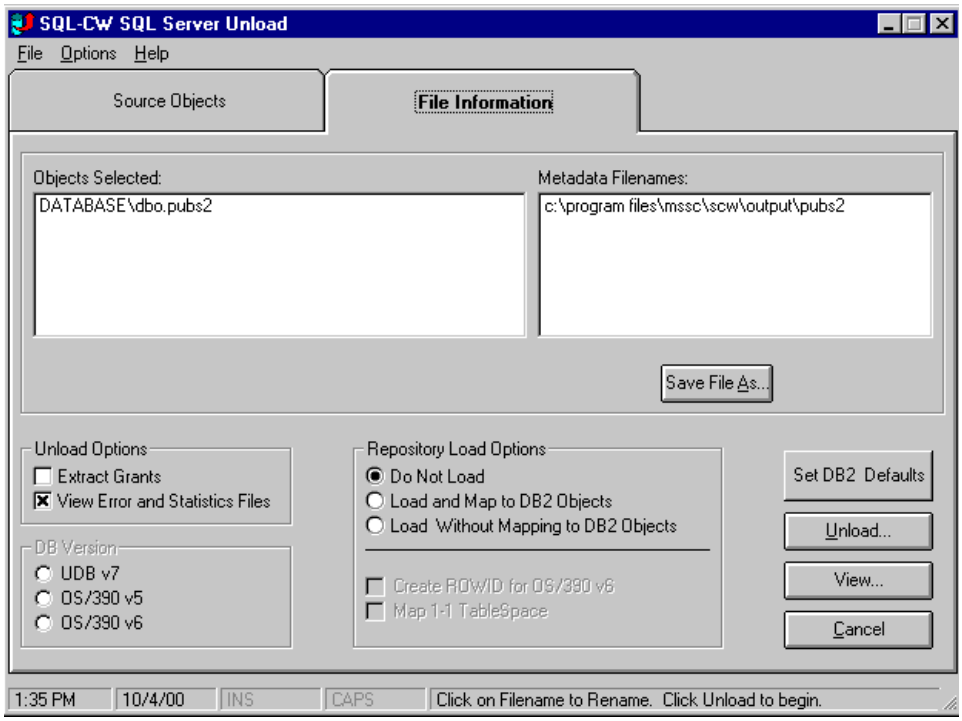

Figure 48. SQL Server unload window

When you have made your selection(s), click the **Unload** button and the data will be unloaded. This may take a few minutes if you have large databases.

When the unload process is completed, you will see a couple of windows, one with any errors listed, and a second with the metrics for the metadata unloaded. The paths for these files are listed at the top of the windows if you

would like to look at the files later. Figure 49 shows the metrics for the metadata unloaded.

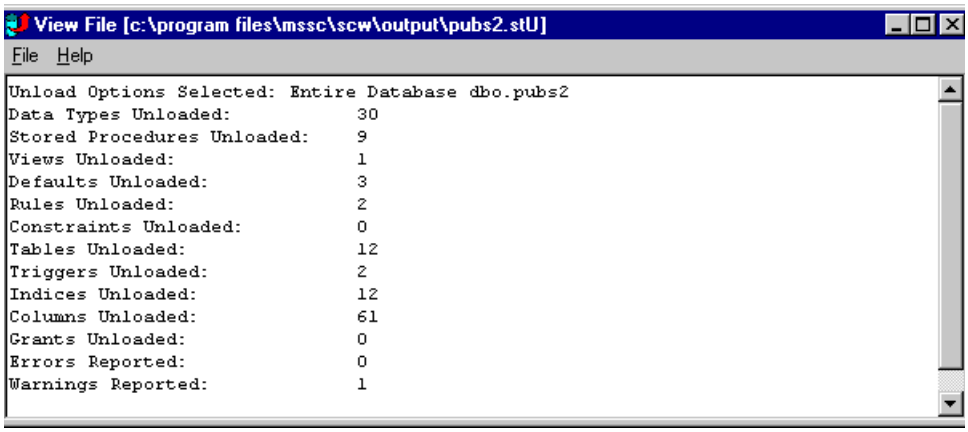

Figure 49. Unloaded metadata

A.1.3 Loading metadata

When your metadata is unloaded with no errors, the next step will be to load the metadata to the repository database. To load the metadata, select **START > Programs > SQL Conversion Workbench > Meta load**, or click the Repository Load icon on the main Window. This will bring up a window as shown in Figure 50.

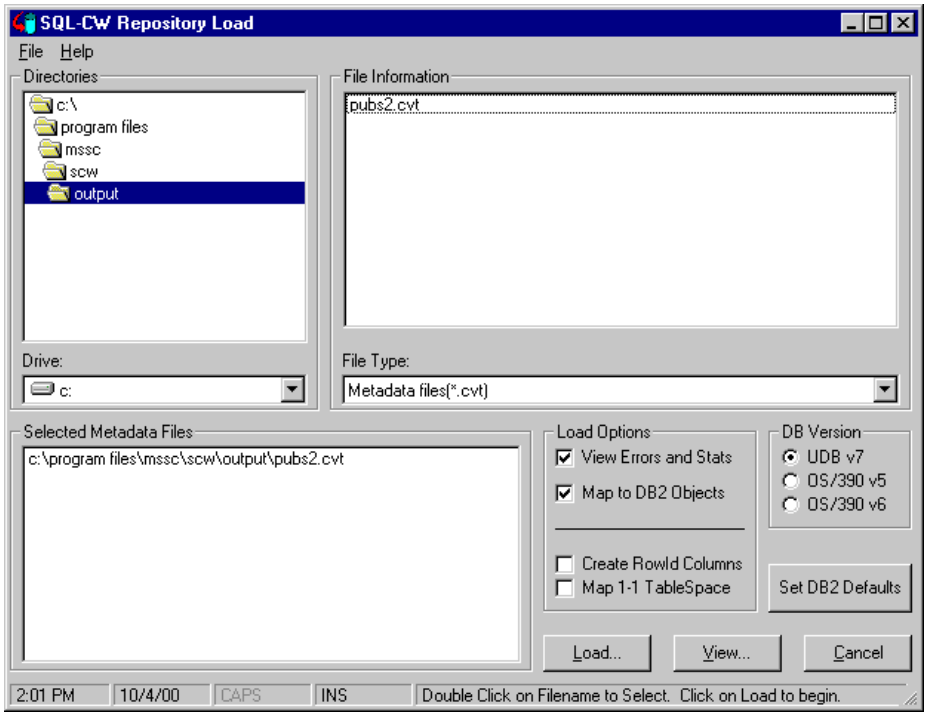

Figure 50. Load metadata to repository

The first thing you will see when you click the **Load** button is a window for login to your repository database (Figure 51):

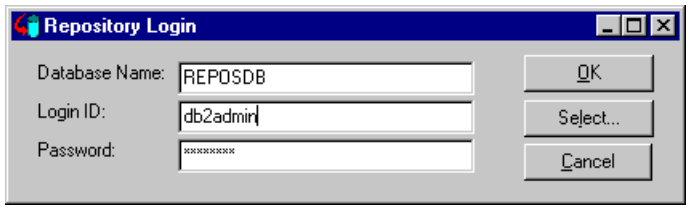

Figure 51. Login to repository database

When the process completes, you will see a list of errors or warnings. Figure 52 is a sample of our error window.

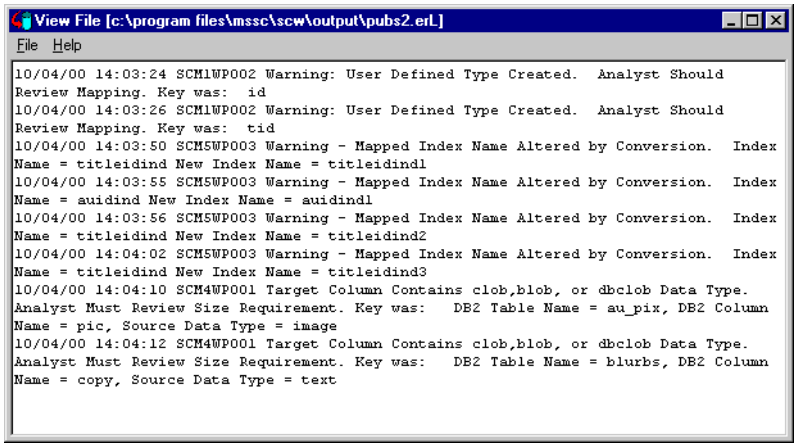

Figure 52. Load repository errors

Note that some of the mapped index name are altered because an index name must be unique within a DB2 database.

An `IMAGE` or `TEXT` data column is mapped to a column with the data type `BLOB(2G)` or `CLOB(2G)` by default. You need to adjust the size based on the required size for these columns.

Now that we have completed loading the table metadata to the repository, our next step will be to edit the data. The first thing we did was to go over the complete list of errors and warnings and correct those.

A.1.4 Editing metadata

The repository editor allows you to perform a number of tasks to prepare your metadata for the new DB2 platform as well as some DB2 administrative tasks. To name a few, you can:

- Customize your target database
- Rename your target database
- Set up and modify table spaces
- Create and modify buffer pools
- Modify tables and columns
- Create, modify or delete foreign keys
- View privileges if you selected the **Extract Grants** option before doing the unload

To edit the metadata you will need to click the **Repository Edit** icon on the main window, or you can select **TOOLS>> Repository Edit** to bring up the edit window.

You will be required to select database from and to, file names to edit as seen Figure 53.

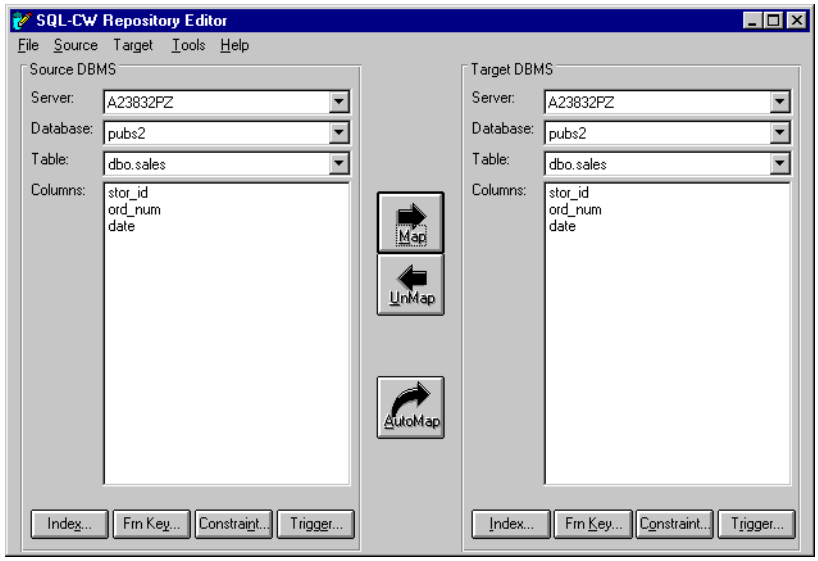

Figure 53. Repository edit window

There are a number of reports that can be obtained regarding errors, list constraints, views, triggers and other types of objects.

When you have completed the editing and are satisfied with the results, you can generate DDL for your target DB2 UDB database.

To generate the DDL you will need to click **File > Generate DDL** and a window as shown in Figure 54 will appear.

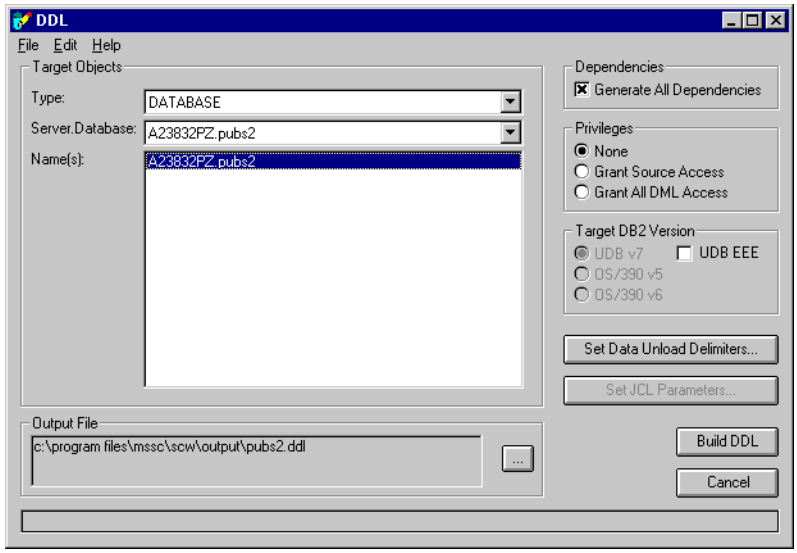

Figure 54. Generate DDL

Here you will need to select your files, server and database that you want to generate DDL for your new DB2 UDB database. Press Build DDL and the process will begin.

There is a button to select your delimiting character for the columns output. The default is the tilde (~), so be sure this character does not exist in your data or select another delimiting character.

When the process completes you will see a small window with the list of files created as shown in Figure 55, the type of files and the path they were written in. You will need to review these files and make any corrections necessary.

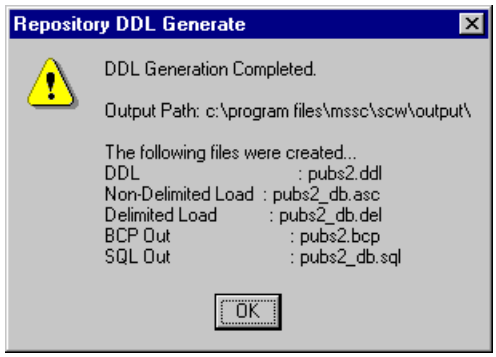

Figure 55. DDL file and scripts created

You will want to look at these files and make corrections if necessary. The files created will allow you to create database, table spaces, tables and other database objects as well as provide both select statements and BCP commands to unload your data. There is also a file containing the necessary commands to load the data to your DB2 UDB database.

You are now ready to build your target database. This can be accomplished by using the DDL file created in the previous step.

Open a DB2 command window and execute the DDL as the following example:

```
db2 -f pubs2.ddl -t -r output.txt -s -l errorlog.txt
```

Check this step for errors and you will be ready to begin the data migration process.

A.1.5 Data migration

Data migration statements were created in the previous step. These may require modification. Pay special attention to the Sybase `datetime` data type and make sure it is in the correct format for the DB2 `timestamp` data type.

A.1.6 Stored procedure conversion

Stored procedure conversion can be accomplished with SQL-CW product in either batch mode or in interactive mode.

A.1.6.1 Batch code conversion

For batch mode, you will select **File > DBMS Procedures**. You will then see a window as shown in Figure 56:

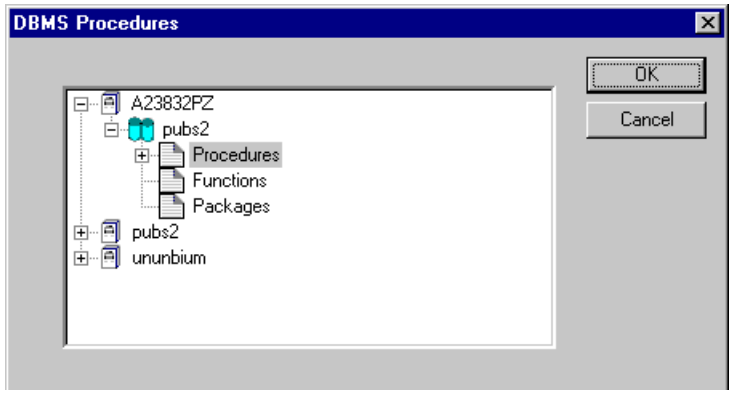

Figure 56. Procedure batch conversion selection window

You then click **Procedures** and click **OK** and the procedure conversion begins. A small window as shown in Figure 57 will appear asking if you want to continue.

Figure 57. Stored procedure conversion

Click OK to begin the conversion process.

There will be a progress bar to indicate status. This may take a few minutes if you have many stored procedures.

This conversion process will create three types of files,

- *.sta files contain assessment metrics
- *.err files contain errors and warnings
- *.sqp files contain the converted stored procedure

Figure 58 is a sample of an error (.err) file:

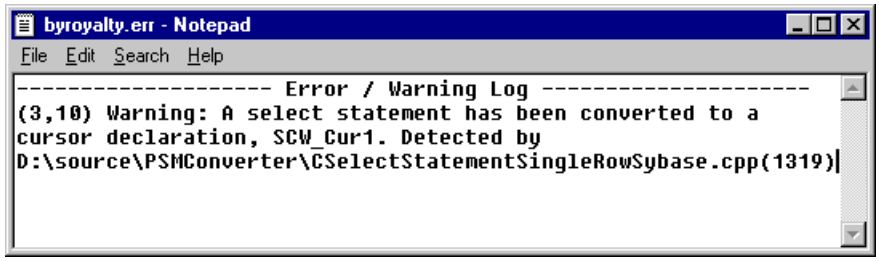

*Figure 58. *.err file sample*

Once the procedures are converted you will need to carefully go through all *.err files to check for errors and warnings and make all necessary corrections.

A.1.6.2 Interactive conversion

The interactive conversion mode allows you to convert individual stored procedures one at a time.

To begin interactive conversion, from the main screen menu bar, select **File > DBMS Procedures** and expand the Procedures folder.

Select the procedure you want to convert and click **OK** and the source stored procedure text will appear as shown in Figure 59:

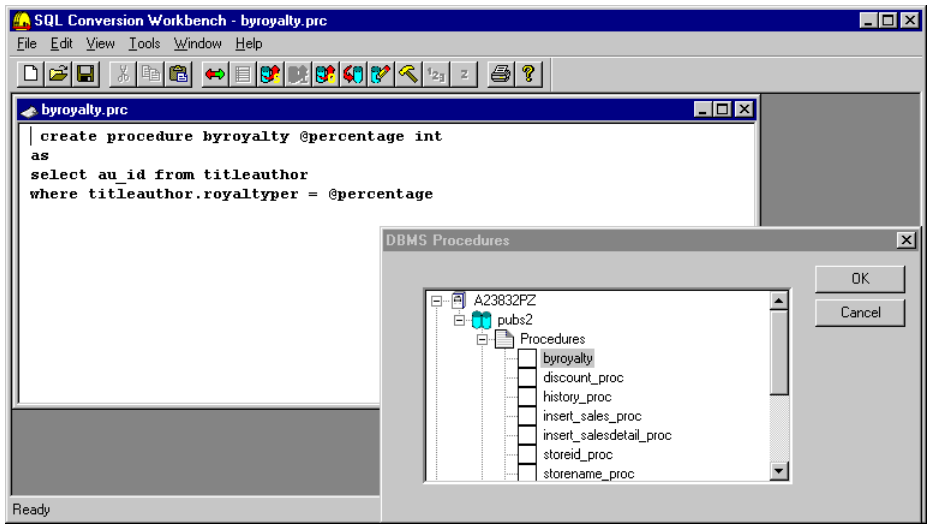

Figure 59. Interactive convert screen

The **Convert** button is activated on the main screen, and when you click it, the stored procedure is converted and the output is displayed as Figure 60.

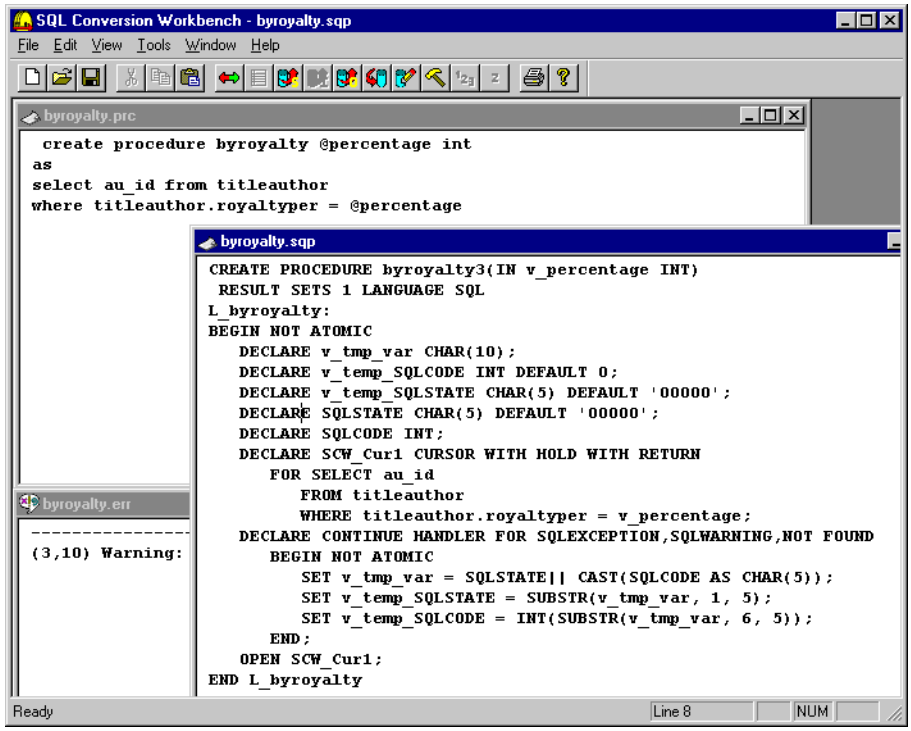

Figure 60. Interactive stored procedure output

When a stored procedure is converted, an error file (.err) is also generated. Go though all the errors and warning and make all necessary corrections.

After you have checked and cleaned up all errors, you can execute the CREATE PROCEDURE statement using DB2 Stored Procedure Builder (SPB) or from the DB2 Command Line Processor (CLP).

A.2 Other tools

There are many tools you may utilize for your conversion project. Here we will introduce two of these tools: Data Junction and Platinum ERwin.

A.2.1 Data Junction

Data Junction offers a transformation tool for DB2 UDB data migration and application integration. It is a visual design tool for building and testing data transformations that work between DB2 and other data formats such as

Sybase Adaptive Server. Sybase SQL Server is supported via a native API, a Massive Insert API, and the `bcp` command.

Projects and transformations designed with Data Junction can be executed by the DJEngine. This tool is an engine that executes data transformations on demand or as scheduled.

The graphic interface allows the definition of source-to-target mapping and transformation rules. It accounts for data type differences, and can set various filters to dynamically modify target columns during the conversion process.

For more information, see the Data Junction Technical Overview published at the following Web site:

```
http://www.datajunction.com/products/dj_technical.html
http://www.datajunction.com/products/matrix_formats_1.html
```

Or visit the Data Junction Corporation Web site at the following URL:

```
http://www.datajunction.com
```

A.2.2 PLATINUM ERwin

ERwin is a database design tool that aids in designing and maintaining database applications. A logical model, along with business rules, defines the database, and a physical model represents the target database. ERwin allows visualization of the structure, key elements, and design of a database. It automatically generates tables, stored procedures, and trigger code for leading databases such as DB2 UDB and Sybase SQL Server.

ERwin can also be used to reverse-engineer database objects using a DDL script or existing database. The physical model allows users to select different target databases and generate DDL script for every target. Using this feature, DDL scripts for different databases and versions can be easily supported.

For more information, see the PLATINUM ERwin Fact Sheet published at the following Web site:

```
http://www.platinum.com/products/factsht/erwin_fs.htm
```

Or visit the PLATINUM Technologies Web site at the following URL:

```
http://www.platinum.com/products/ap_dev.htm
```

Appendix B. Sybase and DB2 UDB terms

This appendix attempts to relate some common tasks and terms. The Sybase task is listed, along with the equivalent DB2 UDB steps to accomplish the same task or similar task.

B.1 Sybase versus DB2 UDB terminology

The first thing is to look at some comparisons of terms relating to administrative tasks. Although they will not relate exactly, they will be close (see Table 34).

Table 34. Sybase versus DB2 UDB terminology

SYBASE	DB2 UDB
Sybase server	DB2 UDB Instance
database	database
device	table space and containers
segment	
login to server	attach to instance[a]
use database	connect to database
master database	system catalogs
tempdb	temporary table spaces

a. You can execute the `ATTACH` command to attach to the DB2 instance on the same workstation or remote workstation. To perform instance administrative tasks like creating a database, updating the database manager, and killing connected database user, you have to attach to the DB2 instance. If you have not executed the `ATTACH` command, all the instance level commands are executed against the current instance, specified by the `DB2INSTANCE` environment variable.

B.2 Sybase versus DB2 task comparison

In Table 35, there are some administrative tasks a DBA would normally preform, along with the Sybase and DB2 UDB solutions. Sybase Central has functionality to perform most of these tasks and the DB2 UDB provides most

of this functionality through the Control Center, the Command Center or the Stored Procedure Builder (SPB).

Table 35. Sybase versus DB2 UDB comparable tasks

Task	Sybase	DB2 UDB
View the server options	sp_configure	GET DBM CFG
View the database options		GET DB CFG FOR *dbname*
Update server options	sp_configure *'option_name', new_value*	UPDATE DBM CFG USING *'config_parameter', new_value*
Update database options		UPDATE DB CFG FOR *dbname* USING *'config_parameter', new_value*
Display active servers or instances	showserver (executed from UNIX shell)	db2ilist (executed from UNIX shell)
Access server or instance	isql -U*user* -P*pswd* -S*server*	ATTACH TO *instance_name* user *user_name* using *pswd*
Access database	use *dbname*	CONNECT TO *dbname* user *user_name* using *pswd*
List databases in server or instance	sp_helpdb	LIST DB DIRECTORY
List devices or files used by the databases	sp_helpdevice	LIST TABLESPACES or LIST TABLESPACE CONTAINERS
Find space used or available space	sp_helpdb *dbname* or sp_helpsegment *segname* or sp_spaceused	LIST TABLESPACE CONTAINERS FOR *(tsid)* SHOW DETAIL or LIST TABLESPACES SHOW DETAIL
List database tables	sp_help	LIST TABLES
List table characteristics	sp_help *tablename*	DESCRIBE TABLE *tablename*

Task	Sybase	DB2 UDB
List source for stored procedures	sp_helptext *procname*	Use 'Get Source' function of the DB2 Stored Procedure Builder
Administer security	grant revoke sp_helpuser sp_addlogin sp_adduser sp_addalias sp_dropalias sp_dropuser sp_droplogin sp_addgroup sp_helpgroup sp_changegroup sp_password	GRANT REVOKE UPDATE DBM CFG USING SYSADM_GROUP *group_name* UPDATE DBM CFG USING SYSCTRL_GROUP *group_name* UPDATE DBM CFG USING SYSMAINT_GROUP *group_name* All the authentication set up is done by an external security mechanism such as Operating System (mkuser, chuser, mkgroup, chgroup, passwd)
Start a server or instance	startserver -f*srvr_name* (from UNIX shell)	db2start (from UNIX shell)
Backup database	dump database *db_name* to '*/path/file*'	BACKUP DATABASE *db_name* TO */path/file*
Restore database	load database *db_name* from '*/path/file*'	RESTORE DATABASE *db_name* FROM */path/file*
Export a text file from a table	bcp *table_name* out *filename* ...	EXPORT TO *filename* OF *type* SELECT ...
Load a text file into a table	bcp *table_name* in *filename* ...	LOAD FROM *filename* OF *type* INSERT INTO *tablename* or IMPORT FROM *filename* OF *type* INSERT INTO *tablename*
List connected users	sp_who	LIST APPLICATIONS
Kill connected users	kill *spid_number*	FORCE APPLICATION
Generate DDLs	defncopy	db2look -e

B.3 Sybase logging versus DB2 UDB logging

Transaction logging is a different concept in Sybase and DB2 UDB. As you know with Sybase, each database has a table `syslogs`, which is used during database recovery. It is up to the Database administrator to control these logs, both by setting options for the `syslogs` table and schedules for dumping the logs. The dump transaction command has options to:

```
dump transaction database_name to '/path/file'
dump transaction database_name with no_log
dump transaction database_name with truncate_only
```

Recovery is accomplished by using the `syslogs` online to the database and if that fails, then you can recover by loading a database and the load transaction as;

```
load database database_name from '/path/file'
load transaction database_name from '/path/file'
```

DB2 UDB has a separate logging strategy and different recovery methodology. Logging is accomplished in circular or archival method.

When using circular logging, the log files are based on the size specified in the `LOGFILESIZ` and the number of files specified in the `LOGPRIMARY` and `LOGSECONDARY` database configuration parameters. Based on these values, DB2 UDB fills a file, then goes to the next until the maximum values of files and size are consumed. When a log file has had all of it's transactions committed, it can be reused. Crash recovery and version recovery are possible using circular logging, but not roll-forward recovery.

When archival logging is used, when all transactions in the log have been committed, it is closed and the next file used. The closed file is now online archived. Based on the values of the `LOGRETAIN` database configuration parameter, such files will remain online archived and a user exit routine can move them to offline archiving, meaning not in the active log file directory. Recovery, including roll-forward recovery can be done with archival logging.

Appendix C. Special notices

This publication is intended to help database administrator and system designers to understand and evaluate the activities involved in performing a database and application conversion from Sybase Adaptive Server for AIX Version 12 to DB2 UDB for AIX Version 7.1. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 UDB for AIX Version 7.1. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 UDB for AIX Version 7.1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been

reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

e (logo)® 
IBM ®
AIX
AS/400
CT
DataJoiner
DB2 Connect
DRDA
OS/2
RS/6000
VisualAge
XT
Redbooks
Redbooks Logo 
AT
DATABASE 2
DB2
DB2 Universal Database
Netfinity
OS/390
RETAIN
S/390
System/390

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

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

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

Sybase, Adaptive Server, Adaptive Server Enterprise, Client-Library, DB-Libaray, Open Client are trademarks of Sybase,Inc. in the United States and/or other countries.

The SQL Conversion Workbench is a trademark of ManTech Systems Solutions Corporation in the United States and/or other countries.

Data Junction is a trademark of Data Junction Corporation in the United States and/or other countries.

Erwin is a trademark of Computer Associates International, Inc. in the United States and/or other countries.

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

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

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.

Appendix D. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

D.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 255.

- *Converting from Oracle AIX to DB2 for OS/390*, SG24-5478
- *DATABASE 2 for AIX Conversion Guide Oracle 7.1 to DB2 Version 2*, SG24-2567
- *DataJoiner Implementation and Usage Guide*, SG24-2566
- *Planning for Conversion to the DB2 Family: Methodology and Practice*, GG24-4445

D.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at ibm.com/redbooks for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

D.3 Other resources

These publications are also relevant as further information sources:

- *DB2 UDB for UNIX, Quick Beginnings*, GC09-2970
- *DB2 UDB Installation and Configuration Supplement*, GC09-2957
- *DB2 UDB SQL Reference, Volume 1,* SC09-2974
- *DB2 UDB SQL Reference, Volume 2,* SC09-2975
- *DB2 UDB Command Reference,* SC09-2951
- *DB2 UDB Data Movement Utilities Guide,* SC09-2955
- *DB2 UDB Application Building Guide,* SC09-2948
- *DB2 UDB Application Development Guide,* SC09-2949
- *DB2 UDB Call Level Interface Guide and Reference,* SC09-2950
- *DB2 UDB Administration Guide: Planning,* SC09-2946
- *DB2 UDB Administration Guide: Implementation,* SC09-2944
- *DB2 UDB Administration Guide: Performance,* SC09-2945
- *DB2 Universal Database V6.1 for UNIX, Windows, and OS/2 Certification Guide,* ISBN:0-13-086755-1
- *Installation Guide Sybase Adaptive Server Enterprise for IBM RISC System/6000 AIX*, 35892-01-1200-01
- *Configuring Adaptive Server for UNIX Platforms*, 35823-01-1200-01
- *Sybase Adaptive Server Enterprise Reference Manual*, 36271-01-1200
- *Sybase Adaptive Server Enterprise Utility Programs for UNIX platforms*, 36124-01-1200-01
- *Sybase Adaptive Server Enterprise Transact-SQL User's Guide*, 32300-01-1200
- *Sybase Adaptive Server Enterprise System Administration Guide*, 32500-01-1200

D.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- http://www.ibm.com/
  IBM Home Page
- http://www.redbooks.ibm.com/
  ITSO Home Page
- http://www-4.ibm.com/software/data/db2/udb/
  DB2 UDB Home Page
- http://www-4.ibm.com/software/data/db2/migration/
  DB2 Migration Home Page
- http://www-4.ibm.com/software/data/db2/library/
  DB2 Product and Service Technical Library
- http://www-4.ibm.com/software/data/db2/udb/winos2unix/support/
  DB2 Universal Database and DB2 Connect Online Support
- http://www.sybase.com/
  Sybase Home page
- http://sybooks.sybase.com/
  Sybase Product Manuals
- http://www.mantech.com/
  ManTech Systems Solutions Corporation Home Page

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** ibm.com/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

Symbols

A

B

C

D

E

F

P

R

S

T

U

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6128-00 DB2 UDB V7.1 Porting Guide
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

DB2 UDB V7.1
Porting Guide

A practical guide to porting Sybase Adaptive Server to DB2 UDB

Techniques and considerations for migration projects

Databases, applications, and data conversion

This IBM Redbook is intended to help database administrators and system designers perform database and application conversion from Sybase to DB2 Universal Database (DB2 UDB) Version 7.1. It contains a description of the conversion process and suggestions on how the mapping of database features may be accomplished.

The target audience of this redbook is database administrators and system designers with background in Sybase and/or DB2 UDB for AIX.

This book was written from the AIX operating system perspective, and some commands shown in this book may not be available on other operating systems. However, all sections in this book except some operating system specific commands and operations should be applicable for database systems on non-AIX operating systems such as Solaris and Windows NT.

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

SG24-6128-00 ISBN 0738419400