DBMaker

DBMaker Tutorial

CASEMaker®

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive

Santa Clara, CA 95050, U.S.A.

www.casemaker.com

www.casemaker.com/support

Document No. 645049-237118/DBM541-M02282017-TUTO

Publication Date: 2017-02-28

For a description of updated functions that do not appear in this manual, read the file named README.TXT after installing the CASEMaker DBMaker software.

Trademarks

CASEMaker, the CASEMaker logo, and DBMaker are registered trademarks of CASEMaker Inc. Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corp. UNIX is a registered trademark of The Open Group. ANSI is a registered trademark of American National Standards Institute, Inc.

Other product names mentioned herein may be trademarks of their respective holders and are mentioned only form information purposes. SQL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

Notices

The software described in this manual is covered by the license agreement supplied with the software.

Contact your dealer for warranty details. Your dealer makes no representations or warranties with respect to the merchantability or fitness of this computer product for any particular purpose. Your dealer is not responsible for any damage caused to this computer product by external forces including sudden shock, excess heat, cold, or humidity, nor for any loss or damage caused by incorrect voltage or incompatible hardware and/or software.

Information in this manual has been carefully checked for reliability; however, no responsibility is assumed for inaccuracies. This manual is subject to change without notice.

Contents

1 Introduction 1-1

1.1 Additional Resources 1-3

1.2 Technical Support 1-4

1.3 Document Conventions 1-5

2 RDBMS Basics 2-1

2.1 Syntax Diagrams 2-3

2.2 RDBMS Functions 2-4

2.3 Data Models 2-6

2.4 Data Independence 2-7

Physical 2-7

Logical 2-8

2.5 High-Level Language Support 2-9

2.6 Transaction Management 2-10

What is a Transaction? 2-10

Concurrency Control 2-11

The Lock Concept 2-11

2.7 Integrity Control 2-14

2.8 Access Control 2-15

User Authorization 2-15
Transaction Authorization 2-15
2.9 RDBMS Recovery 2-17
System Failures 2-17
Media Failures 2-17
3 RDBMS Architecture 3-1
3.1 Logical RDBMS 3-2
Internal or Physical Level 3-3
Conceptual Level 3-3
External or View Level 3-3
Mappings between Levels 3-4
3.2 Physical RDBMS 3-5
Applications and Utilities 3-6
Application Program Interface (API) 3-8
Query Language Processor 3-9
RDBMS Engine 3-10
4 Databases 4-1
4.1 Naming Conventions 4-2
4.2 dmconfig.ini File 4-3
Creating 4-3
Directory 4-4
Format 4-4
Section Names 4-5
Keywords 4-5
Comments 4-6
4.3 dmSQL 4-7
Starting 4-7
Workspace 4-8
4.4 JTools 4-10
JConfiguration Tool 4-10
JServer Manager 4-10

JDBA Tool 4-11
4.5 Creating a Database 4-12
Tutorial Database 4-12
Connection Handles 4-12
Default User 4-13
4.6 Database Modes 4-15
Single-User Mode 4-15
Multiple-Connection Mode 4-15
Client/Server Mode 4-16
5 Tables 5-1
5.1 Tablespaces 5-2
Regular Tablespaces 5-2
Autoextend Tablespaces 5-2
System Tablespace 5-2
Default User Tablespace 5-3
The Temporary Tablespace 5-3
5.2 Data Types 5-5
BIGINT 5-5
BIGSERIAL (start) 5-5
BINARY (size) 5-6
CHAR (size) 5-7
DATE 5-7
DECIMAL (NUMERIC) 5-8
DOUBLE 5-9
FILE 5-9
FLOAT 5-10
INTEGER 5-11
JSONCOLS 5-11
LONG VARBINARY (BLOB) 5-16
LONG VARCHAR (CLOB) 5-16
NCHAR (size) 5-17
NVARCHAR (size) 5-18

OID 5-19
REAL 5-19
SERIAL (start) 5-20
SMALLINT 5-21
TIME 5-21
TIMESTAMP 5-22
VARCHAR (size) 5-23
Media Types 5-23
5.3 Creating a Table 5-25
Default Values for Columns 5-28
Lock Mode 5-32
Fillfactor 5-33
NOCACHE 5-33
Temporary Tables 5-34
6 Data 6-1
6.1 Inserting 6-2
Inserting Using Host Variables 6-3
Different Data Types 6-4
Inserting Blob Data 6-5
6.2 Updating 6-7
Updating Using Standard SQL 6-7
Updating Using Host Variables 6-7
Updating Using OIDs 6-8
6.3 Result Sets 6-9
Selecting Tables 6-9
Selecting Columns 6-11
Selecting Rows 6-12
6.4 Operator Types 6-13
Comparison Operators 6-13
Logical Operators 6-15
Arithmetic Operators 6-16
6.5 Deleting 6-17

Deleting Using Standard SQL 6-17
Deleting Using Host Variables 6-18
Deleting Using OIDs 6-18

7 Database Objects 7-1

7.1 Views 7-2
Creating Views 7-2
Dropping Views 7-3

7.2 Synonyms 7-5
Creating Synonyms 7-5
Dropping Synonyms 7-6

7.3 Indexes 7-7
Creating Indexes 7-8
Dropping Indexes 7-10

8 Users and Privileges 8-1

8.1 Security Management 8-2

8.2 Authority Levels 8-3
Resource 8-3
DBA 8-3
SYSDBA 8-4
SYSADM 8-4

8.3 New Users 8-5
User Access 8-5
Multiple Users 8-6

8.4 Promoting Authority Level 8-7
Multiple Users 8-7

8.5 Demoting Authority Level 8-8

8.6 Removing Users 8-9

8.7 Passwords 8-10

8.8 Managing Groups 8-12
Creating 8-12

Adding Members 8-13
Removing Members 8-13
Dropping 8-14
Nested Groups 8-14
8.9 Table Level Privileges 8-15
Select 8-15
Insert 8-15
Delete 8-15
Update 8-15
Index 8-16
Alter 8-16
Reference 8-16
8.10 GRANT Privileges 8-17
GRANT Table Privileges 8-18
GRANT Column Privileges 8-19
8.11 REVOKE Privileges 8-20
REVOKE Table Privileges 8-21
REVOKE Column Privileges 8-22
9 Database Recovery 9-1
9.1 Types of Failures 9-2
System 9-2
Media 9-2
9.2 Recovery Methods 9-3
Journal Files 9-3
Checkpoint Events 9-3
Recovery Steps 9-4
9.3 Types of Backup 9-6
Full Backup 9-6
Differential Backups 9-6
Incremental Backup 9-7
Offline Backup 9-7
Online Backup 9-8

Online Incremental to Current Backups 9-8
Backup Combinations 9-9
9.4 Backup Modes 9-10
NONBACKUP Mode 9-10
BACKUP-DATA Mode 9-10
BACKUP-DATA-AND-BLOB Mode 9-11
Tablespace BLOB Backup Mode 9-11
Backup Mode of File Objects 9-12
Backup Mode of Stored Procedures 9-13
Compressing Backup Files 9-15
Setting Backup Mode 9-15
9.5 Offline Full Backup 9-20
Offline Full Backup using dmSQL 9-20
Offline Full Backup Using JServer Manager 9-20
9.6 Backup Server 9-22
Starting Backup Server 9-23
Differential Backup Filename Format 9-26
Incremental Backup Filename Format 9-27
Setting Multiple Backup Paths 9-30
Backup Directory 9-31
Setting the Old Directory 9-34
Differential Backup Settings 9-36
Incremental Backup Settings 9-38
Journal Trigger Value Settings 9-41
Compact Backup Mode Settings 9-44
Full Backup Schedule 9-47
Backup Mode of File Objecta 9-49
Backup Mode of Stored Procedures 9-53
Inactivate Backup Server 9-56
9.7 Backup History Files 9-58
Locating the Backup History File 9-58
Understanding the Backup History File 9-58
Using the Backup History File 9-59

Understanding the File Object Backup History File..... 9-59
Understanding the Stored Procedure Backup History File ... 9-60
9.8 Backup on Replication Databases 9-61
9.9 Recovery Options..... 9-63
Analyzing Options 9-63
Preparing for Restoration 9-63
Performing a Restoration..... 9-64
Restoring database by Rollover..... 9-65

1 Introduction

Welcome to the CASEMaker family of products. DBMaker is a powerful and flexible SQL Database Management System (*DBMS*) that supports an interactive Structured Query Language (SQL), a Microsoft Open Database Connectivity (ODBC) compatible interface, and Embedded SQL for C (ESQL/C). DBMaker also supports a Java Database Connectivity compliant interface and DBMaker COBOL interface (DCI). The unique open architecture and native ODBC interface give you the freedom to build custom applications using a wide variety of programming tools or to query databases using existing ODBC-compliant applications.

DBMaker is easily scalable from personal single-user databases to distributed enterprise-wide databases. The advanced security, integrity, and reliability features of DBMaker ensure the safety of critical data. Extensive cross-platform support permits you to leverage existing hardware, allows for expansion and upgrades to more powerful hardware as your needs grow.

DBMaker provides excellent multimedia handling capabilities to store, search, retrieve, and manipulate all types of multimedia data. *Binary Large Objects (BLOBs)* ensure the integrity of multimedia data by taking full advantage of the advanced security and crash recovery mechanisms included in DBMaker. *File Objects (FOs)* manage multimedia data while maintaining the capability to edit individual files in the source application.

This book is intended for end users who are not familiar with DBMaker. It has been written to provide a practical and demonstrative introduction to DBMaker for first-time users. This book presumes you have a general working knowledge of computers,

and are comfortable using the operating system you are using to run DBMaker. Information on the operating system is beyond the scope of this book; consult operating system documentation if you encounter any problems in this area.

This book contains general information on the concepts and principles needed to understand the organization and structure of a database created and maintained using DBMaker. This information presents manageable segments on a single topic. The examples and illustrations provided help you to understand the information presented more clearly.

The SQL language as implemented by DBMaker is covered in this book, and syntax diagrams for commands are provided where the command first appears. The syntax diagrams show you at a glance, possible options, and syntax variations for each command. Explanations of SQL commands include several examples and notes on important points to watch for when using the command.

Most of the concepts, commands, and examples in this book use dmSQL, the command-line tool provided with DBMaker. A few operations can only be performed using other DBMaker application tools or utilities. Refer to Section 1.1, *Additional Resources* for more information on application tools and utilities provided with DBMaker.

1.1 Additional Resources

DBMaker provides a complete set of RDBMS manuals in addition to this one. For more information on a particular subject, consult one of the books listed below:

- For more information on designing, administering, and maintaining a DBMaker database, refer to the *Database Administrator's Guide*.
- For more information on DBMaker management, refer to the *JServer Manager User's Guide.*
- For more information on DBMaker configurations, refer to the *JConfiguration Tool Reference.*
- For more information on DBMaker functions, refer to the *JDBA Tool User's Guide.*
- For more information on the DCI COBOL interface tool, refer to the *DCI User's Guide*.
- For more information on the SQL language used in dmSQL, refer to the *SQL Command and Function Reference*.
- For more information on the SQL language used in dmSQL, refer to the dm*SQL User's Guide*.
- For more information on the ESQL/C programming, refer to the *ESQL/C User's Guide*.
- For more information on the native ODBC API and JDBC API, refer to the *ODBC Programmer's Guide* and *JDBC Programmer's Guide*.
- For more information on error and warning messages, refer to the *Error and Message Reference*.

1.2 Technical Support

CASEMaker provides thirty days of complimentary email and phone support during the evaluation period. When software is registered, an additional thirty days of support will be included, thus, extending the total support period for software to sixty days. However, CASEMaker will continue to provide email support for any bugs reported after the complimentary support or registered support has expired (free of charges).

For most products, support is available beyond sixty days and may be purchased for twenty percent of the retail price of the product. Please contact sales@casemaker.com for details and prices.

CASEMaker support contact information, by post mail, phone, or email, for your area is at: www.casemaker.com/support. We recommend searching the most current database of FAQ's before contacting CASEMaker support staff.

Please have the following information available when phoning support for a troubleshooting enquiry or include this information in your correspondence:

- Product name and version number
- Registration number
- Registered customer name and address
- Supplier/distributor where product was purchased
- Platform and computer system configuration
- Specific action(s) performed before error(s) occurred
- Error message and number, if any
- Any additional information deemed pertinent

1.3 Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and Command Line conventions also have a second setting used with indentation.

CONVENTION	DESCRIPTION
Italics	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but replaced by the actual name. Italics can also be used to introduce new words, and are occasionally used for emphasis in text.
Boldface	Boldface indicates filenames, database names, table names, column names, usernames, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
small caps	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.
➲ **Procedure**	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow.
➲ **Example**	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen.
`Command Line`	Indicates text, as it should appear on a text-delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file.

Figure 1-1 Document Conventions Table

2 RDBMS Basics

If you are not familiar with relational database concepts and principles, or using the SQL language, read this book before any other manuals provided with DBMaker.

Read this book from cover to cover, and complete all of the examples in each chapter provided for creating the tutorial database. We recommended completing each of the examples in the order they appear, omission of examples may produce unexpected results or errors in subsequent examples.

If you are already familiar with databases and only want to try some of the features found in specific chapters, first setup the tutorial database for use with that chapter, then run one of the script files provided with this manual.

One of the most common tasks for computer systems today is storing and managing data. It can include facts, figures, pictures, or multimedia. In general, any collection of information about a particular subject is a database.

Before the widespread use of computers, this information was stored on paper in file folders and filing cabinets. To retrieve some information, you would go to a file cabinet, take a file, and look at the information in the folder. As the collection of information got larger, it became increasingly difficult to retrieve data in a timely manner. Even remembering where to find the information became cumbersome.

When people first began using computers to store information, they stored it in files with a specific, known format. They had to remember where the files were located, and they had to know how to find the data in the file.

To obtain the information from the files, a special program had to be written to retrieve the data. Once this program existed, a user could retrieve the data very quickly. However, if the user decided to change the way the data was stored, or wanted to look for different data, a new program was required.

Programmers in a company's information systems department usually wrote these programs. As the amount of information available on the computer grew, the requests for new and different ways to view the data also increased. Large backlogs of user requests for programs became more common, and there were delays of weeks or even months for the new programs. This led to the need of an independent storage system for data and new methods to retrieve it.

2.1 Syntax Diagrams

Syntax diagrams show the syntax for all SQL commands. These diagrams provide assistance when constructing a statement on the command line, but cannot remember the syntax options. See the example syntax diagram displayed below.

To use the syntax diagram, simply follow the line from start to finish. Any elements of the command that you cannot navigate around are required. Any elements that you can navigate around are optional, but provide additional options or flexibility.

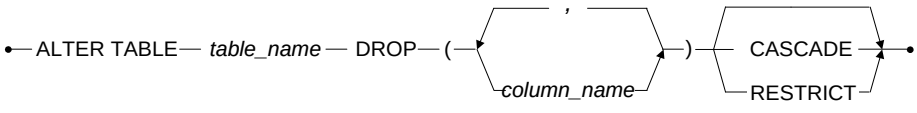


Figure 2-1 Sample Syntax Diagram

Any words that appear in italics are placeholders for the actual names used in a database. Substitute the actual names for these placeholders. In the above diagram, replace the <*table_name*> placeholder with the name of a table in the database. For example, in the tutorial database, replace the <*table_name*> placeholder with **Customers** to execute this command on the **Customers** table.

Also, note the direction of the arrows. Sometimes it is possible to have a list of items in a command, shown in a syntax diagram as a circular path. Both <*column_name*> fields above can include a list of column names, separated by commas, as indicated by the circular path following the arrows.

2.2 RDBMS Functions

A Relational Database Management System manages the data stored in a database. It is a record-keeping system used to maintain information and to make data available upon request. There are important features of DBMaker that make it a powerful and flexible enough system to virtually underlie all other information systems.

A typical Relational Database Management System provides a number of specific functions:

- **Data model** — must have some means of representing the data in a way that a user can easily understand. The data model is actually a mathematical abstraction. It ensures that all of the data present in the database is available and viewable by users.
- **Data independence** — should provide insulation from any physical storage changes in the structure of the database. A request for specific information should return the correct results, even if the physical storage structure of the database has changed.
- **High-level language support** — the information in the database should be accessible by a high-level language. This language should allow a user to define, access, and manipulate data without having to know what the physical storage structure of the database looks like.
- **Transaction management** — should provide some method to ensure that multiple transactions on the same data do not interfere with each other. This allows multiple users to simultaneously use a database.
- **Integrity control** — should guarantee that data in the database does not have invalid values or inconsistency in related data. This prevents a user from accidentally entering invalid data or performing operations that can violate data dependency.

- **Access control** — should provide facilities for protecting the security and privacy of data from unauthorized users. This prevents unauthorized users from gaining access and viewing sensitive data.
- **Recovery methods** — should provide a method for backing up and restoring data in the event of a system failure.

2.3 Data Models

The way data is physically stored on computers probably has little or no significance to a user. All of the data may be stored as simple binary numbers that span several files or even multiply disks. The RDBMS uses a *data model* to represent stored data in a way that is meaningful and easy for a user to understand.

The database model is a mathematical abstraction of data, which provides structural access techniques for the data. This ensures quick manipulation and retrieval of data by users and application programs without the burden of remembering data location or storage methods.

There have been several popular data models over the years, but the relational database model is currently the most widely used. This is also the data model used by DBMaker. The relational database model presents information to the user in a familiar form of tables with rows and columns. Each row contains data on one subject or item, and each column contains attributes, for example name, size and quantity, for these subjects or items.

2.4 Data Independence

One of the biggest advantages of a RDBMS is *data independence*. Data independence allows changes to occur to the structure of a database, without requiring application programs or users to make any changes in the way they access the data. The two kinds of data independence are *physical* and *logical*.

Physical

Early file-based systems stored all of their information in files with a specific format. To retrieve data from the files, a programmer who knew the format of those files had to write a program. If there was a change to the structure of the data, the program had to be changed in order to read the information from the new structure in the proper order. If a user wanted to look at the data with a new view, a new program had to be written. The organization of the data and the access techniques for retrieving that data are built into the application logic and code. This type of system is data dependent.

In a RDBMS, the physical structure of the database may be changed without affecting application programs or altering the user's view of the data. These changes may affect the speed or efficiency of application programs, but the user programs should not have to be altered. This is possible because the RDBMS uses the abstraction provided by the data model to make the physical structure of the database transparent to both users and application programs. The data is translated from the way it is physically stored and accessed on disk to the representation and access techniques used by the outside world, or the logical view.

If the physical structure changes, the RDBMS is aware of these changes and still provides the same logical view. Because the logical view presented to the outside world remains constant, application programs and user interactions based on the logical view of the data do not have to be altered to provide for changes in the physical structure. Therefore, the RDBMS has physical data independence.

Logical

Sometimes it is necessary to make a change in the logical structure of the data. As long as the logical view of the existing data remains the same, this should have no effect on user interactions or application programs. The data model permits the use of abstract characteristics for instance, names to access data instead of the physical characteristics used in a file based system. Since adding data will not alter these abstract characteristics for a data item, no changes in access methods or techniques are required. Existing programs and user queries will run unaffected, and will only have to be modified if the new data must be used. Therefore, the RDBMS has *logical data independence*.

2.5 High-Level Language Support

Most databases usually include the capability to use some type of high-level query language. These high-level languages allow a user to define access and manipulate data without having to reference the physical storage structure of the database.

High-level query language support omits the need to access information in a database by means of writing a program that uses the *Application Program Interface (API).* This low-level access method is very useful for creating user applications that automate common and repetitive tasks, but it does not allow any easy way to do one-time ad-hoc, (unplanned or unexpected), queries. Every time someone wanted to do an ad-hoc query, a program would have to be written to perform the query. This would involve a significant amount of effort and training on the part of users, or would greatly increase the workload for application programmers for a one-time query.

The inclusion of a high-level language makes performing ad-hoc queries a relatively simple task. Most high-level languages supported by databases use an English-like syntax that makes them very easy to learn. High-level query languages are very powerful and are able to perform any functions required of a database. DBMaker uses *Structured Query Language (SQL),* the de-facto standard query language used in the industry today.

2.6 Transaction Management

Database management systems are designed to store a large amount of information and provide simultaneous user access. These users may be performing operations on data simultaneously; some type of transaction management is required to ensure that the correct sequence of data is written to the database.

What is a Transaction?

A transaction is traditionally defined as a logical unit of work, one or more operations on a database that must be completed together to leave the database in a consistent state. A single operation on a database can be a self-contained transaction that must complete successfully and change the data, or fail and leave the data unchanged.

Multiple operations can make up a single transaction. Suppose two kinds of information are stored in a database, records of shipments sent to customers and records of the items currently in stock. When an item is shipped to a customer, it is added to the shipments list. This is one operation on the database. However, the quantity of the item shipped must also be subtracted from the items currently in stock.

If both of these are not completed together, the database will be in an inconsistent state. The quantity of items in stock will be too high; items shipped, but not subtracted from items in stock, or too low; items subtracted from items in stock, but not shipped. Both of these operations together make up a single transaction, and must complete successfully or both will fail.

If a transaction completes successfully and changes the data, we say the transaction has been *committed*. If it fails and leaves the data unchanged, we say it has been *rolled back*.

Concurrency Control

There are usually multiple users that require access to data simultaneously, and delays may result in decreased productivity. As a result, most databases support *concurrent access*. This allows multiple users to access the database simultaneously.

This is does not present a problem if the users are accessing different data, but can become a problem when they operate on the same data. When two user transactions operate on the same data without any coordination, the results become unpredictable. Some transactions may read obsolete data, or modifications that were apparently completed successfully may be lost.

To prevent these types of events from occurring, transactions are *serialized*. Two transactions that are executed concurrently will give the same results as if they were performed one after the other, and each user can access the database transparently. Sometimes one transaction must wait for another transaction to finish using a data item. If the second transaction were allowed to proceed without coordination, the results would still be unpredictable.

Suppose one transaction modifies a data item and then continues to perform other operations. While the other operations are being performed, a second transaction modifies the same data item and continues. Before either transaction can be committed, the first transaction encounters an error and is rolled back. The RDBMS returns the database to the state it was in before the transaction occurred, and gives the data item its original value. The second transaction has not yet been committed; the value it placed in the data item is lost.

To permit transactions to be serialized and prevent uncoordinated access to the database, some form of concurrency control is required. Forms of concurrency control often used by a RDBMS are *locks*.

The Lock Concept

A lock on a data item permits a RDBMS to guarantee a transaction will have exclusive access. No other transaction can perform operations on that item while locked. In a typical multi-user RDBMS, this is not always a practical approach.

Instead, a more complex model is used:

- Different types of locks exist like *share locks* and *exclusive locks*
- Different levels of locks exist like *row locks*, *page locks*, and *table locks*

TYPES OF LOCKS

A share lock allows multiple transactions to access a data item simultaneously, but with one restriction; the transactions cannot modify the data item. This may occur when multiple transactions want to read the value of a data item, but will not change it. In this case, multiple accesses are acceptable because they will not interfere with each other.

When a transaction wants to modify a data item, allowing other transactions to read or modify the data item at the same time can lead to inconsistencies in the database. When a transaction wants to modify a data item, an exclusive lock is used to prevent other transactions from accessing the data. This allows the transaction to continue with its other operations, certain that the data item will remain in a stable state for the duration of its cycle.

A RDBMS may also use different levels of locks, although the reasons behind this are more for performance issues than concurrency control. In a relational RDBMS, the smallest data item that can usually have a lock placed on it is the row. These rows are grouped together forming pages, which are further grouped to produce tables. In a RDBMS, pages and tables can be locked as a single item, locking all data items contained within.

LOCK ESCALATION

If a transaction has to access many rows in a page, the time needed to acquire all of the individual locks and resources used to keep track of items would be quite long. Using a page lock would reduce the time and resources used, but at the cost of concurrency for other transactions. If a second transaction wants to acquire a lock on one of the rows in the same page, it will now be unable to do so. However, this is usually offset by the gain in performance.

A similar situation occurs when a transaction accesses many pages in a table, using a table lock instead of a page lock will decrease the time and resources used at the expense of concurrency. The level of lock used on a specific data item can be set manually in the transaction. A RDBMS can use an *automatic lock escalation* when it determines that performance will be improved, still maintaining an acceptable level of concurrency by automatically changing a lock to the next higher level.

2.7 Integrity Control

People assume that data contained in a database is accurate. This is the primary reason database systems have evolved, to provide the ability to retrieve accurate data in a timely manner. To ensure this is true, a RDBMS must have some form of integrity control. Integrity control ensures that the data is consistent and valid.

Inconsistency can result when there is redundancy in the database, such as when the same data exists in two separate places in the database and the RDBMS is not aware of the duplication. It would be possible for a transaction to update only one of the two entries. After that, the RDBMS could supply incorrect or contradictory information to users, and clearly be in an inconsistent state. A RDBMS with integrity control will generate an error if this occurs in a properly designed database.

It is also possible to retain the redundancy in the database, as long as it is controlled. In this case, the RDBMS is aware of both entries, and any change to one will cause the RDBMS to update the other one as well. This is generally referred to as a *cascading update*. It is possible for the database to temporarily be in an inconsistent state during the update procedure, the RDBMS would make the data unavailable to users until the update is finished.

Ensuring data is valid is also an important function of the database. A payroll database that shows an employee worked 400 hours in a week instead of 40 clearly contains invalid data. There is no way for the RDBMS to determine if this value is invalid itself, but a RDBMS with the proper integrity control functions can allow the Database Administrator to define and implement *integrity constraints*. These integrity constraints will check to ensure that data is valid whenever a transaction attempts to modify data.

2.8 Access Control

The centralized and multi-user nature of a RDBMS requires that some form of security control be in place, to prevent unauthorized access and to limit access for authorized users. Security control can generally be divided into two areas, user authorization, and transaction authorization.

User Authorization

User authorization protects a database against unauthorized use, usually by requiring that a user enter a username and a password to gain entry to the system. The password is usually known only to the user and the RDBMS, and is protected by the system. The username and password scheme cannot guarantee the security of a database. Tell users to choose a password that is not easy to guess and does not contain personal information like the name of a spouse or a pet. Users should never write or post their password in an insecure location, like on the front of a computer!

Transaction Authorization

Generally, users are not given the same access rights to a database. Sensitive data such as employee salaries should only be accessible to users who need it. In other cases, users may only require the ability to read some data items, where other users require the ability to both read and update the data.

A Point-Of-Sale (POS) system is a good example. Clerks at a store might need read-access for item prices, but should not be able to change the price. Employees at the head office may need to read and update the data to enter new prices or items.

Transaction authorization helps to protect a database against an authorized user intentionally or unintentionally trying to access a data item, they do not have permission to access. The RDBMS keeps a record of what rights have been granted to users on all data objects, and checks these rights every time a user's transaction tries to access a database. If the user does not have the proper rights to a data item, the

transaction will not be allowed. It is the responsibility of the Database Administrator to explicitly grant rights to each user.

2.9 RDBMS Recovery

A RDBMS may fall victim to a software or hardware failure. Failures are divided into two types, system failures, and media failures. A method for recovery after a failure is one of the main advantages a RDBMS has over a file-based system.

System Failures

A system failure occurs when the *volatile storage* fails in a computer system. Volatile storage is the term used for the main memory in a computer system. A system failure may be caused by a power failure, a program/operating system crash, or some other reason. The most common method of protecting against system failures is the use of a transaction journal or transaction log.

The transaction journal is a history of all changes made to the database. The exact state of a transaction in progress cannot be reliably determined in the event of a system failure and it cannot be completed when the system restarts. The RDBMS uses the transaction journal to undo all changes that have been written to disk for transactions that terminated abnormally.

It is possible that a transaction have completed before the system failure, but not all changes have been written to disk. The data may still be stored in the RDBMS system buffers at the time of the failure. In this case, the RDBMS uses the transaction journal to redo or *rollover* all transactions.

Media Failures

Media failure occurs in a disk storage system. Media failures are usually caused by physical trauma to the disk itself, such as excessive heat or a head crash resulting from exposure to vibration or g-forces outside its physical operating limits. There is nothing to prevent the loss of data on the affected disk. However, the database can be restored if the database provides archiving or data mirroring.

Archiving is a backup of a database at periodic intervals, every night for example. This allows saving a backup copy of every file transaction that has occurred since the last backup. When a media failure occurs, use the backup copies to reconstruct a database up to the point in time of the last backup. All changes made since the last backup will be lost. This type of archiving is suitable for some database systems, but is not robust enough for critical applications such as electronic banking or airline reservation systems.

Data mirroring involves continuously creating an archive copy of the entire database. A copy of the entire database at a single point in time and a duplicate transaction journal are created. Any changes made to the database are written to both logs simultaneously, in effect creating two copies of the database. If a media failure occurs during a log time, then only the part that was not written to the second log will not survive. There is still a record in the other log, an error message can be sent to the user notifying them of the loss during recovery.

When using either of these methods, store the backup copies in a location away from the original database to ensure its survival for restoration.

3 RDBMS Architecture

There are two different ways to look at the architecture of a RDBMS: the logical RDBMS architecture and the physical RDBMS architecture. The logical architecture deals with the way data is stored and presented to users, while the physical architecture is concerned with the software components that make up a RDBMS.

3.1 Logical RDBMS

The logical architecture describes how users perceive data in the database. It is not concerned with how the data is handled and processed by the RDBMS, but only with how it looks. The way data is stored on the underlying file system is transparent to the user. Users can manipulate the data without worrying about where it is located or how it is actually stored. This results in the database having different levels of abstraction.

The majority of commercial Database Management Systems available today are based on the *ANSI/SPARC* generalized RDBMS architecture, as proposed by the ANSI/SPARC Study Group on Data Base Management Systems. The ANSI/SPARC architecture divides the system into three levels of abstraction: the internal or physical level, the conceptual level, and the external or view level.

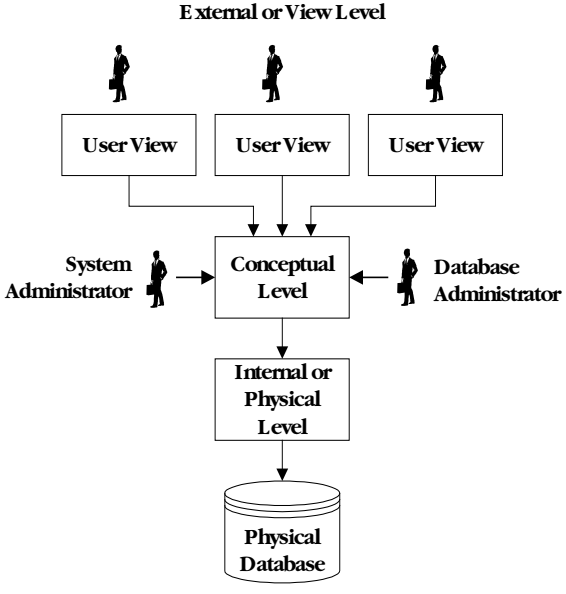


Figure 3-1: The logical architecture of a typical RDBMS

Internal or Physical Level

The collections of files permanently stored on secondary storage devices are the physical database. The physical or internal level is closest to the physical storage. It provides a low-level description of the physical database and an interface between the operating system's file system and the record structures used in higher levels of abstraction. It is at this level that record types and methods of storage are defined. It also defines; how stored fields are represented, what physical sequence the stored records are in, and what other physical structures exist.

Conceptual Level

The conceptual level presents a logical view of the entire database as a unified whole, which brings all data in the database together for viewing in a consistent manner. The first stage in the design of a database is to define the conceptual view; a RDBMS provides a data definition language for this purpose.

The conceptual level allows a RDBMS to provide data independence. The data definition language used to create the conceptual level must not specify any physical storage considerations that should be handled by the physical level. It should not provide any storage or access details, but should only define the information content.

External or View Level

The external or view level provides a window on the conceptual view, which allows the user to see only data of interest to them. The user can be either an application program or an end user. Any number of external schemas can be defined and overlap each other.

The System and Database Administrators are special cases. Because they have responsibilities for the design and maintenance of a database, they need to be able to see the entire database. The external and conceptual views are functionally equivalent for these two users.

Mappings between Levels

The three levels of abstraction in a database do not exist independently of each other. There must be some correspondence or mapping between the levels. There are actually two mappings, the conceptual/internal mapping, and the external/conceptual mapping.

The conceptual/internal mapping lies between the conceptual and internal levels, and defines the correspondence between records and fields in the conceptual view, and the files and data structures of the internal view. If the structure of the stored database changes, then the conceptual/ internal mapping must also change. It is this mapping that provides physical data independence for the database.

The external/conceptual view lies between the external and conceptual levels, and defines the correspondence between a particular external view and the conceptual view. Although these two levels are similar, some elements found in a particular external view may be different from the conceptual view. For example, several fields may be combined into a single (virtual) field, which can also have different names from the original fields. If the structure of the database at the conceptual level changes, then the external/conceptual mapping must change, accordingly so the view from the external level remains consistent. It is this mapping that provides logical data independence for the database.

It is also possible to have another mapping, where one external view is expressed in terms of other external views; this could be called an external/external mapping. This is useful if several external views are closely related to one another. It allows a user to avoid mapping each of the similar external views directly to the conceptual level.

3.2 Physical RDBMS

The physical architecture describes the software components used to enter and process data, and how these software components interconnect. At its most basic level, the physical RDBMS architecture can be broken down into two parts, the back end, and the front end.

The back end is responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels. Other benefits of a RDBMS, such as security, integrity, and access control, are also the responsibility of the back end.

The front end consists of any application that runs on top of the RDBMS. These may be applications provided by the RDBMS vendor, the user, or a third party. The user interacts with the front end, and may not even be aware that the back end exists.

The back end and front end can be further broken down into the software components that are common in most types of RDBMS.

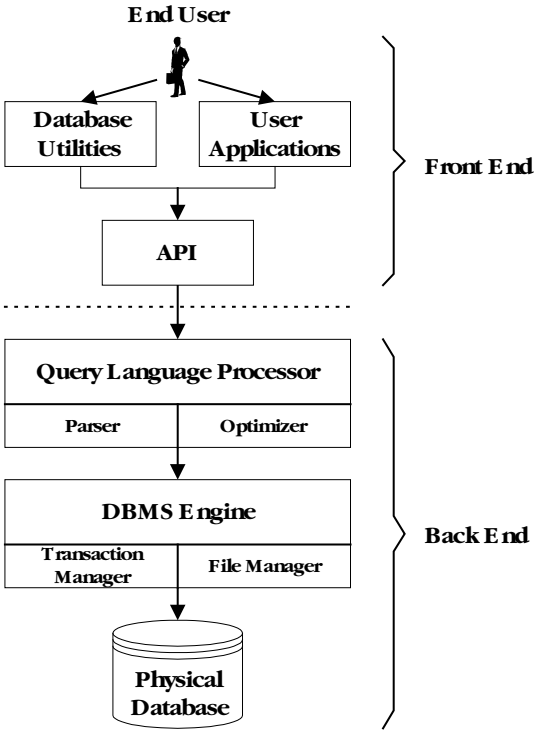


Figure 3-2: Common functions and components of a RDBMS

Applications and Utilities

Applications and utilities are the main interface to the RDBMS for most users. There are three main sources of applications and utilities for a RDBMS a vendor, a user, and third parties.

VENDOR APPLICATIONS

Vendor applications and utilities are provided for working with or maintaining the database, and allow users to create and manipulate a database without the need to write custom applications. These are usually general-purpose applications and are not the best tools to use for doing specific, repetitive tasks.

USER APPLICATIONS

User applications are custom-made application programs written for a specific purpose using a conventional programming language. This programming language is coupled with the RDBMS query language through the API. This allows the user to utilize the power of the RDBMS query language with the flexibility of a custom application.

THIRD PARTY APPLICATIONS

Third party applications may be similar to those provided by the vendor with enhancements, or they may fill a perceived need that the vendor has not included with the application. They can also be similar to user applications, written for a specific purpose they think a large majority of users will need.

APPLICATIONS AND UTILITIES LIST

The most common applications and utilities used with a database fall into several well-defined categories:

Command Line Interfaces

Character-based, interactive interfaces that directly use the full power and functionality of a RDBMS query language. They allow manipulation of a database, perform ad-hoc queries, and see the results immediately. They are often the only method of exploiting the full power of a database without creating programs using a conventional programming language. DBMaker includes the command line interface dmSQL.

Graphical User Interface (GUI) tools

Graphical, interactive interfaces that hide the complexity of the RDBMS and query language behind an intuitive, easy to understand, and convenient interface. This allows casual users the ability to access the database without having to learn the query language, and it allows advanced users to quickly manage and manipulate the database without the trouble of entering formal commands. Graphical interfaces usually do not provide the same level of functionality as a command line interface because it is not always possible to implement all commands or options. DBMaker includes three main GUI tools: JConfiguration Tool, JServer Manager and JDBA Tool.

Backup/Restore Utilities

Designed to minimize the effects of a database failure and ensure a database is restored to a consistent state when a failure does occur. Manual backup/restore utilities require the user to initiate the backup, while automatic utilities will backup the database at regular intervals without any intervention from the user. Proper use of a backup/restore utility allows a RDBMS to recover from a system failure correctly and reliably.

Load/Unload Utilities

Allow the user to unload a database or parts of a database and reload the data on the same machine, or on another machine in a remote location. This can be useful in several situations; creating backup copies of a database at a specific point in time, or for loading data into a new version of the database, or into a completely different database. These load/unload utilities may also be used for rearranging the data to improve performance, such as clustering data together in a particular way or reclaiming space occupied by data that has become obsolete.

Reporting/Analysis Utilities

Used to analyze and report on the data contained in a database. This may include analyzing trends in data, computing values from data, or displaying data that meets some specified criteria, and then displaying or printing a report containing this information.

Application Program Interface (API)

The application program interface (API) is a library of low-level routines, which operate directly on the database engine. The API is usually used when creating software applications with a general-purpose programming language, for instance C++ or Visual Basic. This allows a user to write custom software applications to suit the needs of a business, without having to develop the storage architecture. The database engine handles the storage of the data. The input and any special analysis or reporting functions are handled by the custom application.

An API is specific to each RDBMS, and a program written using the API of one RDBMS cannot be used with another RDBMS. Each API usually has its own unique function calls that are tied very tightly to the operation of the database. Even if two databases have the same functionality, they may use different parameters and functions, depending on how the database was designed. One exception to this is the Microsoft Open Database Connectivity API, designed to work with any RDBMS that supports it.

Query Language Processor

The query language processor is responsible for receiving query language statements and changing them from the English-like syntax of the query language to a form the RDBMS can understand. The query language processor usually consists of two separate parts, the parser, and the query optimizer.

PARSER

The parser receives query language statements from application programs or command-line utilities and examines the syntax of the statements to ensure they are correct. The parser breaks a statement down into basic units of syntax and examines them to make sure each statement consists of the proper component parts. If the statements follow the syntax rules, the tokens are passed to the query optimizer.

QUERY OPTIMIZER

The query optimizer examines the query language statement, and tries to choose the most efficient way of executing the query. The optimizer will generate several query plans to perform operations in different orders, and then try to estimate which plan will execute most efficiently. The query optimizer may examine CPU time, disk time, network time, sorting methods, and scanning methods, when making the estimate.

RDBMS Engine

The RDBMS engine is the heart of the RDBMS, and it is responsible for all data management. The RDBMS engine usually consists of two separate parts, the transaction manager and the file manager.

TRANSACTION MANAGER

The transaction manager maintains tables of authority and concurrency controls. The RDBMS may use authorization tables to allow the transaction manager to ensure the user has permission to execute the query language statement on the database. The authorization tables can only be modified by authorized user commands, which are checked against the tables of authority. A database may also support concurrency control tables to prevent conflicts when simultaneous, conflicting commands are executed. The RDBMS check the concurrency control tables before executing a query language statement to ensure that it is not locked by another statement.

FILE MANAGER

File manager is responsible for a database's physical input/output operations. It is concerned with the physical address of data on a disk and is responsible for any interaction, reads or writes with the host operating system.

4 Databases

With DBMaker, a user can easily create and manage a database. Extensive cross-platform support and a unique open architecture allow the user to deploy a database application across several platforms, and easily migrate to larger systems as the system grows. Easily scale from a small single-user database on a notebook computer all the way to a large multi-user database distributed around the world.

This book is intended to instruct you primarily on using the command-line utility dmSQL to perform database management functions. DBMaker also provides JTools; graphical utilities that simplify database management.

In this chapter you will learn:

- How to choose a valid name for a database
- How to partition data
- How to start the dmSQL command-line tool
- How other DBMaker utilities can be used to perform database management routines
- How to create a database
- How to configure a database
- How to start and terminate a database
- How to connect to and disconnect from a database

4.1 Naming Conventions

With DBMaker, it is possible to have several databases running on a single computer at the same time. In order to tell DBMaker which database to connect to, you need some way to identify one database from another. DBMaker does this using a naming scheme.

DBMaker stores configuration information for all local and remote databases in the **dmconfig.ini** file, using the same name for two different databases will cause a conflict. DBMaker will not be able to tell which of the configuration sections is for which database, and may write the configuration information in the wrong place. Check the section headings in the **dmconfig.ini** file to see the database names that already exist and choose a new unique database name.

Carefully choose a name from one to one hundred and twenty-eight characters in length to use before executing the CREATE DB command. A database name can contain letters, numbers, and the underscore character and cannot be changed once created.

➲ **Example**

```
Tutorial
Parts_db
Region 1
1_Region
```

Database names are not case-sensitive. This means that **Tutorial** can be entered when creating the database and users can logon using **tutorial** and **TUTORIAL**. **Tutorial** is the name of the database that will be used throughout this book.

4.2 dmconfig.ini File

DBMaker stores all configuration information for each database, including the database name, in a file called the **dmconfig.ini** file. This file contains a database configuration section for each database that a user can connect. The **dmconfig.ini** file is a regular ASCII text file, and can be edited with any text editor.

DBMaker also provides a GUI Tool, the JConfiguration Tool, to simplify maintenance of the **dmconfig.ini** file. Its descriptive interface reduces the time needed to become familiar with DBMaker configuration parameters, and organizes the parameters into clearly defined categories.

In most cases, DBMaker looks at the configuration information when a database starts. If you change this information after starting a database, it will not take effect until the next time the database starts. However, there are some configuration parameters that are only required when connecting to a database. You can change this information anytime before connecting to the database, and the new values will be used when the next connection is made.

The configuration parameters play an important role in the performance of DBMaker. You should be aware of the effects of each configuration parameter and estimate the best values to use to ensure DBMaker will run smoothly. Refer to the *Database Administrator's Reference* for a full description of the configuration parameters and the keywords DBMaker uses to control them.

Creating

A user should not normally need to create a new **dmconfig.ini** file, since the DBMaker installation program will automatically create one. When creating a database, DBMaker will examine the **dmconfig.ini** file and look for a configuration section name that corresponds to the name of the new database. If it finds a matching configuration section, it will then check for any creation-time configuration options, and use the values when it creates the new database.

A user should create the database configuration section with a text editor before creating a database, if any creation-time options use a value other than the default value, so the parameters will take effect when creating the database. If DBMaker cannot find a configuration section in **dmconfig.ini** while creating a database, it will automatically create the section in the first **dmconfig.ini** file it finds or in a new **dmconfig.ini** file if it cannot find an existing one. When DBMaker creates a new configuration section, it uses the default values for all creation-time configuration options. In general, the default values for the creation-time configuration options should be fine for most databases.

Directory

On Windows systems, the **dmconfig.ini** file is in the **Windows** directory. When using Windows systems, a user can also open the **dmconfig.ini** file from the Start menu. Click the Start button, point to *Programs* > *DBMaker 5.4*> *DBMaker Configuration File (**dmconfig.ini**)*.

On UNIX systems, the **dmconfig.ini** configuration file can be in one of three locations. When starting a database, DBMaker will scan these three locations in the order listed below to locate a **dmconfig.ini** file with a section name that corresponds to the database:

1. The current directory
2. The directory specified in the environment variable of DBMAKER
3. DBMaker's installation directory: **~dbmaker*Version***

If a **dmconfig.ini** file and the section name are found, the keywords defined in that section will be used. If the section name cannot be found in that file, DBMaker will continue searching for a **dmconfig.ini** file in the next directory until successful.

Format

The **dmconfig.ini** file is divided into sections called database configuration sections. Every database has its own database configuration section, and the values in that section control the configuration operations for that database.

Each database configuration section is made up of a section header followed by one or more keyword lines. The section header is the name of the database enclosed in square brackets. The keyword lines consist of a keyword and a corresponding value or values.

➲ **Example**

```
[section header 1]
keyword1 = value1                ;text following a semicolon is a comment
keyword2 = value2
         .
         .
[section_header_2]
keyword3 = value3 value4         ;spaces or commas may be used
keyword4 = value5                ;as delimiters between values
         .
         .
```

Keywords in the **dmconfig.ini** file are not case-sensitive. Values may or may not be case-sensitive, depending on the keyword and the operating system the database is running on.

Section Names

The name of each section corresponds to the name of the database that will use the configuration options found in that section when it starts up. The section name begins with a left square bracket ([) followed by the name of the database, and ends with a right square bracket (]). The brackets are required to enclose the section name, and the left bracket must be the first character on the line.

Keywords

Following each section name is a list of keywords and their values. These values will be used by the database that corresponds to the section heading when it starts. The statement "*keyword* = *value*" assigns the specified value to a keyword. If a keyword requires or supports multiple values, separate individual values with either spaces or commas. The value can be an integer or a string.

If DBMaker cannot find a keyword in **dmconfig.ini**, it will use a default value. Depending on their purpose, keywords may be used at either start time or connect

time. For a complete list of keywords and their values, refer to the *Database Administrator's Guide*, Appendix B.

Comments

Any string or symbol that is written after the semi-colon (;) is considered a comment and ignored by DBMaker. You can use comments to remind a user what a keyword is for, why you chose a specific value for a keyword, or what the original value for a keyword was if changing it temporarily.

4.3 dmSQL

dmSQL is a character-based, interactive user interface that utilizes the full power and functionality of the SQL query language found in DBMaker. Use dmSQL to manipulate the database, perform ad-hoc SQL queries, and to see result sets immediately. dmSQL is often the only method of exploiting the full power of a database without creating programs using a conventional programming language.

Starting

Most of the examples in this tutorial use dmSQL, you need to know how to start this application, and should familiarize yourself with the program before using it. You will learn how to configure a new database for client/server operation later in this chapter.

WINDOWS

On Windows platforms, the functionality of both dmsqls and dmsqlc has been combined into a single program, dmsql32.exe.

➲ **To start the dmSQL command-line utility in Windows98 or WinNT:**

1. Click the **Start** button
2. Select **Programs,** then select DBMaker, and click **dmSQL**
3. The **dmSQL** application starts

UNIX

The UNIX versions of DBMaker provide a single-user version (dmsqls) and a client/server version (dmsqlc) of the dmSQL application. Use dmsqls to create a database in UNIX for a single-user or client/server database.

➲ **To start the dmSQL command-line utility in UNIX:**

1. At the command line, type:

```
cd ~DBMaker/<current version>/bin
```

NOTE *Substitute <current version> for the version of DBMaker, e.g., 5.4.*

2. Press ENTER
3. Type the following command on the command line:
```
dmsqls
```
4. Press ENTER
5. The dmSQL application starts

Workspace

After starting dmSQL, the dmSQL workspace displays on Windows systems, or the dmSQL> command-line prompt on UNIX systems.

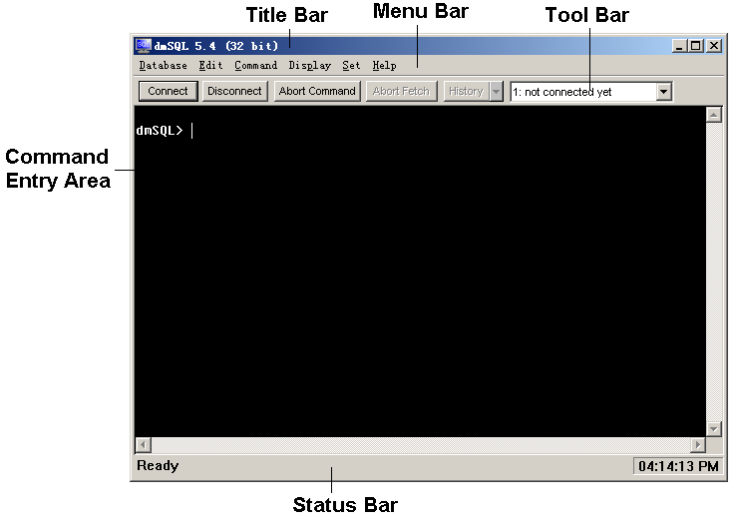


Figure 4-1: A Windows version of dmSQL

The dmSQL window contains the following areas:

- **Title Bar**—The title bar displays the program name ("dmSQL") and the **Minimize**, **Maximize**, and **Close** buttons
- **Menu Bar** — The menu bar displays dmSQL pull-down menu titles. Each menu contains a list of related commands

- **Toolbar** — The toolbar is a palette of command buttons and drop-down list boxes for many commonly used functions
- **Command Entry Area** — The command entry area is the main window in the dmSQL workspace to enter commands, and where dmSQL runs scripts and displays text.
- **Status Bar** — The status bar describes the current activity in the workspace, and displays the current time.

4.4 JTools

DBMaker provides three Java-based, cross-platform utilities for managing databases. The tools provide an easy-to-use, intuitive interface and are designed to quickly familiarize you with the database. Each of the JTools comes with its own documentation and content-sensitive on-line help. JTools can be used on any operating system that supports ODBC. The following sections introduce these tools and their functions, but further discussion is beyond the scope of this book. For more information, refer to each tool's respective documentation.

JConfiguration Tool

As mentioned in section 4.2 *dmconfig.ini File*, the JConfiguration tool manages the database's configuration parameters. JConfiguration tool provides a descriptive interface that allows you to configure a database without needing to remember the meanings of keywords in the **dmconfig.ini** file and their possible values. On Windows operating systems, you can start the JConfiguration Tool by clicking Start>Programs>DBMaker 5.4>JConfiguration Tool. For detailed instructions on how to use JConfiguration Tool, refer to the *JConfiguration Tool Reference* or the help provided with the tool.

JServer Manager

JServer Manager provides a simple graphical interface for performing the most common database management routines: creating, starting, shutting down, deleting, backing up, and restoring databases. Using JServer Manager is beyond the scope of this book, however, examples of how to backup and restore a database using JServer Manager are included in Chapter 9, *Database Recovery*. JServer Manager is highly recommended to help simplify these routines. On Windows operating systems, you can start the JServer Manager by clicking Start>Programs>DBMaker 5.4>JServer Manager. For detailed instructions on how to use JServer Manager, refer to the *JServer Manager User's Guide* or the help provided with the tool.

JDBA Tool

JDBA Tool gives a clear and functional view of the logical organization of an individual database. Database administrators may use it to create, drop, and alter schema objects. On Windows operating systems, you can start the JDBA Tool by clicking Start>Programs>DBMaker 5.4>JDBA Tool. For detailed instructions on how to use JDBA Tool, refer to the *JDBA Tool User's Guide* or the help provided with the tool.

4.5 Creating a Database

Creating the database is probably the simplest part of designing and implementing a database in DBMaker. To create the Tutorial database, simply type the **CREATE DATABASE** command with **Tutorial** being the database name.

Tutorial Database

➲ **To create the Tutorial database using dmSQL:**

1. Type the following at the dmSQL command prompt:
```
CREATE DATABASE Tutorial;
```
2. Press ENTER.
3. The following line displays on the screen:
```
USE db #1 connected to db:<Tutorial> by user:<SYSADM>
```

The command creates an empty database named **Tutorial**. Before it creates the database, it looks in the **dmconfig.ini** file to see if a section with the name already exists. If it does already exist, DBMaker will use the keyword values for the creation-time configuration options from that section to create the new database.

In the **Tutorial** example, you did not create a database configuration section before creating the database; DBMaker will create one and use the default values for all creation-time configuration options.

Connection Handles

A database can have as many as eight simultaneous connections using dmSQL. DBMaker uses the term USE to indicate which database connection is currently active.

This USE is also known as a connection handle. There are eight connection handles, from USE#1 up to USE#8. The first database connection you make is on USE#1, the second on USE#2, and all the way up to USE#8. To view all currently connected databases in dmSQL, enter the USE command at the dmSQL command line.

USE COMMAND

After executing this command, dmSQL will display a list of all databases currently connected. In this case there is only one database so there is only one connection handle (USE#1). This indicates that the database created named **Tutorial** has one handle, USE#1, and a user connected with the username **SYSADM**.

➲ **To view all currently connected databases using dmSQL:**

1. Type USE at the dmSQL command prompt:
```
dmSQL> USE;
```
2. Press ENTER
3. The following displays:
```
dmSQL> USE;
USE db #1 connected to db:<TUTORIAL> by user:<SYSADM>(CURRENT)
```
4. The following shows more than one database connection:
```
dmSQL> use;
USE db #1 connected to db:<TUTORIAL> by user:<SYSADM>(CURRENT)
USE db #2 connected to db:<DBSAMPLE> by user:<SYSADM>
USE db #3 connected to db:<EXDM35> by user:<SYSADM>
```

In this example, you can see that the currently connected database is **Tutorial**, indicated by the text (CURRENT) after the connection information.

To connect to a database with a connection handle other than USE#1, change to a USE# to connect to before making the connection. To change to another connection handle in dmSQL, enter the USE command followed by the number of the connection handle at the dmSQL command line.

➲ **To change to connection handle number two using dmSQL:**

1. Type USE 2 at the dmSQL command prompt:
```
dmSQL> USE 2;
```
2. Press ENTER

Default User

When creating a database, a user automatically connects with the **SYSADM** username. The **SYSADM** has the highest authority in a database and can create new

user accounts, has all rights, and privileges on all database objects. However, since you just created your database, there are not any database objects in the database.

When creating a new database it automatically starts in single-user mode, which only allows one user connection at a time. This allows the opportunity to change the SYSADM password from the default value (no password). If the SYSADM password is not changed, anyone could use the SYSADM account to connect to the database and gain full control over it. Details on changing the SYSADM password will appear in a later section.

To allow other users to connect, run the database in one of the multiple-user modes. This can be single-user mode on UNIX, multiple-connection mode on Windows, or client/server mode on Windows and UNIX. To run the database in multiple-connection mode, terminate the database and then restart it. To run it in client/server mode, terminate the database and add some additional keywords in the **dmconfig.ini** file.

➲ **To terminate the Tutorial database using dmSQL:**

1. Type TERMINATE DATABASE at the dmSQL command prompt:
```
TERMINATE DATABASE;
```
2. Press ENTER.

4.6 Database Modes

DBMaker can start your database in several different modes. Each mode provides different options for connecting to and accessing a database, adding the ability to up-scale the database from a simple single-user system on one computer to a large multi-user system distributed across several computers.

The database modes available depend upon the platform the database server runs on, and how you want to connect. DBMaker has three database modes, single-user, multiple-connection, and client/server.

Single-User Mode

Single-User mode is only available on UNIX or Linux platforms. This is a simplified version of DBMaker for non-sharable databases. The main advantages of this mode are the smaller application size and faster execution speed for most database operations; locks, security, and network support are not required for a single user database. A limitation of this mode is that only one connection can be made to the database at a time and the database cannot run any of the extra servers or daemons; backup server, replication server, or global transaction server. The database is not available over the network so a user must access the database from the host machine. There is no special configuration required for using single-user mode on UNIX, other than remembering that there is no security.

Multiple-Connection Mode

Multiple-Connection mode is only available on the Windows platform. One advantage of this mode is multiple connections to a database, with the full range of security and reliability features of DBMaker. Similar to single-user mode, all connections must access the database from the host machine, since there is no network support. A limitation of this mode is that the database does not support any of the extra servers or daemons: backup server, replication server, or global transaction server.

There is no special configuration required for using multiple-connection mode on a Windows platform, other than remembering that there is no network support.

Client/Server Mode

Client/Server mode is available on all platforms. This mode permits multiple connections to a database from any computer connected to the host computer via a TCP/IP network, and provides the full range of security, reliability, and concurrency control features of DBMaker. In addition, data sent across the network can be encrypted for additional security. This mode supports all of the extra servers and daemons; backup server, replication server, and global transaction server. Client/server mode requires some configuration to work properly. To run a database you must be connected to a TCP/IP network, and have TCP/IP network protocol support installed on all computers that will be used to connect to the database.

CHANGES TO DMCONFIG.INI

After making the changes to the **dmconfig.ini** file, restart the database in client-server mode with the DBMaker server and then connect with a client application, such as dmSQL.

Add these lines to the [**Tutorial**] section in the **dmconfig.ini** file for Client/server mode:

```
[TUTORIAL]
DB_DbDir=C:\DBMAKER\TUTORIAL\DATABASE
DB_UsrId=SYSADM
DB_SvAdr=127.0.0.1
DB_PtNum=54321
```

DB_SvAdr

The **DB_SvAdr** keyword is required on both the client and server sides when running a database in client/server mode. This keyword specifies the IP address of the computer that will be acting as the server for the database. You should replace the number shown above with the IP address of your computer. Note, if your operating system is using Domain Name System (DNS), you may use the DNS name of the server you have installed DBMaker on in place of the IP address.

DB_PtNum

The **DB_PtNum** keyword is required on both the client and server sides when running a database in client/server mode. This keyword specifies the port number the server will listen for connection requests on.

STARTING CLIENT/SERVER

Use the DBMaker server to start a client/server database. The DBMaker server starts the database and waits for database clients, such as dmSQL, to connect. After a client connects, it accepts commands and returns the results. Only a user with DBA authority or higher can start a database. When running the DBMaker Server, provide the name of the client/server database, a username, and password.

Use the CONNECT command to connect to a database. The command works for single-user, multiple-connection, and client/server databases, but remember to start the database first when using a client /server. The CONNECT command has three parameters: the database name, the username, and the user password. Use the SYSADM username with no password for now.

Use the DISCONNECT command to disconnect from a database in Windows and the TERMINATE DB command in UNIX. The commands work for single-user, multiple-connection, and client/server databases. The commands do not have any parameters, they simply disconnect from the database on an active connection handle.

Windows

➲ **To start a client/server database in Windows:**

1. Click the **Start** button
2. Select **Programs** then select DBMaker, and click **DBMaker Server**
3. The DBMaker Server application starts and displays the **Start Database** dialog
4. In the **Database Name** box, select **TUTORIAL**
5. In the **Username** box, type **SYSADM**
6. Type the following command at the dmSQL command prompt:

```
dmSQL> CONNECT TO TUTORIAL SYSADM;
```

7. Press ENTER

8. To close the database, at the dmSQL command prompt type:
```
dmSQL> DISCONNECT;
```

9. Press ENTER

UNIX

➲ **To start dmServer in UNIX:**

1. At the command line, type:
```
$ cd ~DBMaker/<current version>/bin
```

NOTE *Substitute <current version> for the DBMaker version number, e.g., 5.4.*

2. Press ENTER.

3. The current directory changes to the: *~dbmaker/version number/bin*.

4. At the command line, type:
```
$ dmserver -u SYSADM TUTORIAL
```

5. Press ENTER

6. The DBMaker Server starts and runs the **Tutorial** database

7. The following message appears:
```
DBMaker (current version number)
Copyright 1995-2016 CASEMaker Inc. All rights reserved.
SQL Server bound to port 54321
The database has started successfully.
Database Server is running in the background mode.
Process ID = 28030
```

8. At the command line, type:
```
$ dmsqls
```

9. Press ENTER

10. The **dmSQL** application starts

11. Type the following command at the dmSQL command prompt:
```
dmSQL> CONNECT TO TUTORIAL SYSADM;
```

12. Press ENTER

13. To close the database, at the dmSQL command prompt type:
```
dmSQL> TERMINATE DB;
```

14. Press ENTER

5 Tables

The database exists and is ready for use, but there is no place to store data yet. Think of this as an empty filing cabinet; without file folders, there is no place to store information. In the database, you need to create tables before storing information.

Before creating tables, consider where to locate them and what type of data will be placed in them.

5.1 Tablespaces

A DBMaker database can be partitioned into several logical areas of storage known as *tablespaces*. This allows the database to be divided into manageable areas for logical reasons; tables contain related data, or physical reasons; data must be placed on different disks. This groups data or splits data between different physical disks to speed up access time.

Tablespaces can be either fixed in size or automatically extensible. Tablespaces that are fixed in size are called *regular tablespaces*, and tablespaces that can have their size automatically extended are called *autoextend tablespaces*. DBMaker also has special tablespaces called the *system tablespace* and the *Default User tablespace*.

Regular Tablespaces

A regular tablespace has a fixed size and contains one or more data files. A file too small to hold all data that you wish to store in it can be enlarged manually or alternatively another file added. A regular tablespace can have up to 32,767 data files, provided the total number of data pages in all files is 8 TB or less. A regular tablespace can be changed to an autoextend tablespace.

Autoextend Tablespaces

An autoextend tablespace will grow to contain data in a file as it is added. An autoextend tablespace can be converted to a regular tablespace if you don't wish the tablespace to expand any further or when it is reaching the 8 TB size limit. By default, new tablespaces are autoextend tablespaces. The initial size of a data file in an autoextend tablespace will be the number of pages specified in the **dmconfig.ini.**

System Tablespace

All DBMaker databases contain an autoextend system tablespace. Whenever a database is created, DBMaker generates a system tablespace to record the *system catalog*

tables. The system catalog tables are managed by DBMaker and contain detailed information and statistics about everything stored in the database. Other tables cannot be stored in the system tablespace.

Default User Tablespace

All DBMaker databases also contain an autoextend default tablespace. Whenever creating a database, DBMaker generates an empty tablespace to store user tables. All tables created are stored there by default. Specify the tablespace directory when creating a table to be stored within another tablespace.

The Temporary Tablespace

The temporary tablespace（TMPTABLESPACE）is only used to store external temp tables(ETT). The temporary tablespace also is an auto-extend tablespace. It have exactly two types of files: data files and BLOB files. Data files' logical name is DB_TMPDB, and the physical name is DB_TMPDIR/DBNAME.TDB; BLOB files' logical name is DB_TMPBB, and the physical name is DB_TMPDIR/DBNAME.TBB.

When users call "create temporary table" or "select into" statement, ETTs will be generated and stored into "TMPTABLESPACE ". Users can create temp tables in TMPTABLESPACE(of course system will default store ETT in TMPTABLESPACE), but users can't create any permanent table in TMPTABLESPACE. Users can do "ALTER TABLESPACE TMPTABLESPACE SET AUTOEXTEND OFF/ON;" and "ALTER DATAFILE *DB_TMPDB*/ *DB_TMPBB* ADD *n* PAGES;" ,but users can not add files to TMPTABLESPACE or drop files from TMPTABLESPACE TMPTABLESPACE will be created when a database is created, and the size will be reset to default size when the database is started up.

- Users can't create temporary tables in any other tablespace which is not TMPTABLESPACE

- Users can't create any permanent tables which is not ETT in TMPTABLESPACE.
- Users can't add files to TMPTABLESPACE and drop files from TMPTABLESPACE
- Users can't drop TMPTABLESPACE.

5.2 Data Types

Choose a data type when defining a field in a table. Choosing the wrong data type can waste space, or make the application program take extra steps to convert data into a usable form. DBMaker supports 23 different data types.

BIGINT

The BIGINT data type is an exact signed numeric data type with a precision of nineteen and a scale of zero. The BIGINT data type uses 8 bytes of storage and has a maximum value of 9,223,372,036,854,775,807 and a minimum value of –9,223,372,036,854,775,808.

If attempting to move a value larger than the permitted maximum value from a data type such as BIGINT or INTEGER, DBMaker displays a conversion error and does not move the data.

➲ **Example 1**

```
37654
```

➲ **Example 2**

```
857823
```

BIGSERIAL (start)

The BIGSERIAL data type is a special data type that provides a sequence of consecutive values. DBMaker allocates an integer number for each table contained in a database and uses those numbers to generate a unique sequence for the corresponding table. DBMaker manages and maintains these integer numbers internally. The value of each integer value is automatically increased by one each time it is used.

Providing an integer value for the optional START parameter when defining a BIGSERIAL column can specify the first value in a number sequence, or the START parameter omitted to use the default value of ***1***. Each table in a database can have only one column with the BIGSERIAL data type.

The internal value used to generate a BIGSERIAL number is actually an integer value; the BIGSERIAL data type shares all of the properties of the BIGINT data type. It is an exact signed numeric data type with a precision of 19 and a scale of 0, which occupies 8 bytes of storage. The BIGSERIAL data type also has the same range of values as the BIGINT data type, with a maximum value of *9,223,372,036,854,775,806* and a minimum value of *-9,223,372,036,854,775,808*.

Place a NULL, or empty value in the BIGSERIAL column when inserting a new row to insert a sequential number into a BIGSERIAL column. DBMaker will insert the sequential number for that table into the BIGSERIAL column of the new record, and automatically increase the internal value by one.

If inserting a new column, and supplying an integer value for the BIGSERIAL instead of a NULL or empty value, DBMaker will use the supplied integer value instead of the next sequential number; the internal value will not be incremented by 1. If the supplied integer value is greater than the last sequential number generated, DBMaker will reset the sequence of generated sequential numbers to start with the supplied integer value.

➲ **Example 1**

```
10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007
```

➲ **Example 2**

```
10000, 10001, 5000, 10002, 10003, 11000, 11001, 11002
```

BINARY (size)

The BINARY data type is a fixed-length data type that can contain any binary value. The minimum length of BINARY columns is 1 byte and the maximum length is 3992 bytes. Enter a value for the size parameter when creating a BINARY column. Any data entered in a BINARY column shorter than the column length is padded with a zero-value byte.

Enter character data by enclosing the data in single quotes (' '), the same as when entering CHAR data. However, in BINARY columns the data is stored as hexadecimal values representing the ASCII code of the characters, not as the actual characters entered.

Alternatively, enter hexadecimal values directly by enclosing them in single quotes and appending the 'x' character (' 'x) to indicate the string contains a hexadecimal value. It requires two digits to represent all possible values for each byte in hexadecimal; use an even number of digits when entering values.

➲ **Example 1**

```
'AaBbCcDdEe'x
```

➲ **Example 2**

```
'41614262436344644565'x
```

CHAR (size)

The CHAR data type is a fixed-length data type that can contain any character from the keyboard. CHAR columns can have a minimum length of 1 byte, and a maximum length of 3992 bytes. Enter a value for the size parameter when creating a CHAR column.

Any CHAR data in a column that is shorter than the column length is padded with spaces. When entering CHAR data, enclose it in single quotes (' '). Double-byte characters occupy two bytes. If using double-byte characters, account for this when specifying the length of the column.

➲ **Example 1**

```
'This is a CHAR string.'
```

➲ **Example 2**

```
'This is another CHAR string.'
```

DATE

There are two types of DATE data; DATE literal and DATE constant. Date literal represents the present date. DATE constant is a set point in time. The DATE data type is a fixed-length that contains the calendar date (year, month, and day). The DATE data type uses 4 bytes of storage. Valid values for the year are from 0001 to 9999.

The DATE data type has multiple input/output formats. If the values in the database do not appear correctly, or you are not able to enter dates you think are valid, check the date input/output formats to ensure that they are correct.

- **Example 1a**

```
'0001/01/01'
```

- **Example 1b**

```
'0001/01/01'd
```

- **Example 1c**

```
DATE '0001/01/01'
```

- **Example 2a**

```
'1999/12/31'
```

- **Example 2b**

```
'1999/12/31'd
```

- **Example 2c**

```
DATE '1999/12/31'
```

DECIMAL (NUMERIC)

The DECIMAL data type is an exact signed numeric value with a variable precision and scale. Precision refers to the total number of digits in the mantissa, both to the left and to the right of the decimal point. The default value for precision is *17* with a maximum value of = 38. Scale refers to the number of digits to the right of the decimal point. The default value for scale is *6*.

The amount of storage used by a DECIMAL column is based on the actual value entered, not on the default precision and scale values or the precision and scale values entered when defining the column.

To calculate the amount of storage, use the following formula:

$$\#\,\text{of bytes} = \frac{p+1}{2} + 2$$

For example, the number 9283.83 would be stored as 6 bytes.

The actual calculation used is:

$$\#\text{ of bytes} = \frac{p+1}{2} + 2$$

$$= \frac{6+1}{2} + 2$$

$$= 5.5$$

If you attempt to move a value larger than the allowed maximum from a data type such as FLOAT or DOUBLE, DBMaker displays a conversion error and does not move the data. The DECIMAL data type may be abbreviated as DEC.

➲ **Example 1**

```
3452.8373645
```

➲ **Example 2**

```
736.383732652
```

DOUBLE

The DOUBLE data type is an approximate signed numeric data type with a mantissa of precision 15. Precision refers to the total number of digits in the mantissa, both to the left and to the right of the decimal point. The DOUBLE data type uses 8 bytes of storage and has a valid input range from 1.0E308 to –1.0E308.

The smallest valid input values are **1.0E-308 and –1.0E-308.**

➲ **Example 1**

```
2.89837457884451E285
```

➲ **Example 2**

```
-1.93873634847372E-174
```

FILE

The FILE data type is a structured data type that occupies 48 bytes of storage. This data type is similar to the CLOB and BLOB data types and stores the contents of any existing file as an external file that DBMaker can reference the same as any other data.

DBMaker stores the data externally as a file instead of internally as an object. This allows third-party tools to access and manipulate the data in its native format, without having to re-import the data to register any changes in the database. A file object has a maximum path length of 255 characters.

The FILE column stores a reference to a record in the system catalog tables. The system catalog contains information that the database uses to find the file object. When you display a FILE column, you do not actually see what is stored in the FILE column itself. Instead, DBMaker shows one of three views of information stored in the system catalog or the file itself the filename, the file size, or the file contents.

The FILE data type can store data in two ways, as a system file object or as a user file object. A system file object copies an existing file to the file object directory of the database and gives it a unique name. The database manages this file, and deletes it when there are no references to it in the database. A user file object creates a link to an existing file, while leaving the file in the original location with the original name. Since, the user created this file; it will not be deleted when there are no references made to it in the database. DBMaker must have the read permission on a file before you can insert it into the database as a user file object.

When multiple records reference the same file, DBMaker will store only a single copy of the file and share it between records to save disk space. However, from the user's point of view, there is always a dedicated file for each record. DBMaker transparently generates a new file when updating a shared file. Other records sharing that file are not changed, and other users still see the original file. This prevents any changes made to a file in one record from influencing any other records.

FLOAT

The FLOAT data type is an approximate signed numeric data type having a mantissa with a precision of 15. Precision refers to the total number of digits to the left and to the right of the decimal point. The default FLOAT data type uses 8 bytes of storage and has a valid input range from 1.0E308 to –1.0E308. The default FLOAT type can be specified as REAL or DOUBLE with the keyword DB_FLTDB.

The smallest valid input values are **1.0E-308 and –1.0E-308.**

➲ **Example 1**

```
2.89837457884451E285
```

➲ **Example 2**

```
-1.93873634847372E-174
```

INTEGER

The INTEGER data type is an exact signed numeric data type with a precision of 10 and a scale of 0. The INTEGER data type uses 4 bytes of storage and has a maximum value of 2,147,483,647 and a minimum value of -2,147,483,648.

If you attempt to move a value larger than the allowed maximum from a data type such as DOUBLE, DBMaker displays a conversion error and does not move the data. The INTEGER data type may be abbreviated as INT.

➲ **Example 1**

```
393848
```

➲ **Example 2**

```
-298376
```

JSONCOLS

JSONCOLS Type is a column set of dynamic columns. DBMaker supports dynamic columns. A dynamic column does not exist in the table definition, and it's the keys which can be derived from the JSON string and can be used only when a table has declared a column as JSONCOLS column. For details of a dynamic column, please refer to chapter *Using Dynamic Columns* in D*atabase Administrator's Guide*. For details of a JSONCOLS column, please refer to chapter *Using JSONCOLS Type* in *Database Administrator's Guide*. Dynamic columns of a table are stored as JSONCOLS type which is derived from LONG VARBINARY.

➲ **Example 1**

Creating a table that has JSONCOLS type:

```
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
```

or

```
dmSQL> CREATE TABLE student(name CHAR(30));
dmSQL> ALTER TABLE student ADD COLUMN info JSONCOLS;
```

Inserting data into table **student** by using the name of the JSONCOLS type:

```
dmSQL> INSERT INTO student(name,info) VALUES
('jessia','{"desk_id":3,"birthday":"1986-09-19","score":90}');
1 rows inserted
dmSQL> INSERT INTO student(name,info) VALUES
('pine','{"desk_id":4,"birthday":"1987-03-03","score":95}');
1 rows inserted
```

Query table **student** by using "SELECT *":

```
dmSQL> SET blobwidth 80;
dmSQL> SELECT * FROM student;
       NAME                              INFO
================= ================================================


jessia             {"score":90,"birthday":"1986-09-19","desk_id":3}
pine               {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

Query table **student** by using the name of the JSONCOLS type:

```
dmSQL> SELECT name, info FROM student;
       NAME                              INFO
================= ================================================


jessia             {"score":90,"birthday":"1986-09-19","desk_id":3}
pine               {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

Updating data of table **student** by using the name of the JSONCOLS type:

```
dmSQL> UPDATE student SET info = '{"desk_id":7, "birthday":"1986-09-
19","score":88}' WHERE name='jessia';
1 rows updated
```

Modifying data type of the column named **birthday** to DATE:

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN birthday DATE;
```

```
dmSQL> SELECT info FROM student;
                                INFO
=====================================================================

{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES ('mike','8','1985-
02-15','92');
dmSQL> SELECT info FROM student;
                                INFO
=====================================================================

{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

Creating a text index on the JSONCOLS column named student:

```
dmSQL> CREATE TEXT INDEX idx_stu ON student(INFO);
```

Creating a view on the JSONCOLS column named student:

```
dmSQL> CREATE VIEW view1 AS SELECT info FROM student;
dmSQL> SELECT * FROM view1;
                                 INFO
=====================================================================

{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

➲ **Example 2**

The following operations base on table **student**. For details of table **student**, please refer to Example 1.

Inserting data into table **student** by using the names of the dynamic columns:

```
/* implicit data conversion is closed by default */
dmSQL> INSERT INTO student(name,score) VALUES(?,?);
dmSQL/Val> 'demi','85';    /* it is ok */
1 rows inserted
dmSQL/Val> 'finly',82;     /* INT cannot be converted to CHAR */
ERROR (9629): value list syntax error
dmSQL/Val> END;
dmSQL> SET itcmd ON;
dmSQL> INSERT INTO student (name,score) VALUES(?,?);
dmSQL/Val> 'finly',82;     /* using implicit data conversion */
1 rows inserted
dmSQL/Val> END;
dmSQL> SET itcmd OFF;
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('linda','1','1982-
01-01','91');
1 rows inserted
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('glow','2','1984-
03-25','93');
1 rows inserted
dmSQL> INSERT INTO student (name,desk_id,birthday,score)
VALUES('kitty','abc','1980-02-27','97');
1 rows inserted
```

Query table **student** by using "SELECT *":

```
dmSQL> SELECT * FROM student;
     NAME                            INFO
============= ============================================


jessia         {"score":88,"birthday":"1986-09-19","desk_id":7}
pine           {"score":95,"birthday":"1987-03-03","desk_id":4}
mike           {"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
demi           {"SCORE":"85"}
finly          {"SCORE":"82"}
linda          {"BIRTHDAY":378662400000,"DESK_ID":"1","SCORE":"91"}
glow           {"BIRTHDAY":448992000000,"DESK_ID":"2","SCORE":"93"}
kitty          {"BIRTHDAY":320428800000,"DESK_ID":"abc","SCORE":"97"}
8 rows selected
```

Query table **student** by using the names of the dynamic columns:

```
dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID      BIRTHDAY              SCORE
==========  ==========  ==========  ====================
jessia          7           19*           88
pine            4           19*           95
mike            8           19*           92
demi                NULL    NU*           85
finly               NULL    NU*           82
linda           1           19*           91
glow            2           19*           93
kitty          abc          19*           97
8 rows selected
```

Updating/deleting data of table **student** by using the names of the dynamic columns:

```
dmSQL> UPDATE student SET score='88' WHERE name='linda';
1 rows updated

dmSQL> DELETE FROM student WHERE desk_id='2';
1 rows deleted
```

Adding description of dynamic columns to this table:

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN desk_id INT;
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN score DOUBLE;
```

Inserting data into table **student**:

```
dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jane','12','1982-05-
07',96);
ERROR (6150): [DBMaker] the insert/update value type is incompatible with column
data type or compare/operand value is incompatible with column data type in
expression/predicate

dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jim',8,'1984-09-
26',98);
1 rows inserted

dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID       BIRTHDAY            SCORE
==========  ==========  ==========  ====================
jessia                  7  1986-09-19    8.80000000000000e+001
pine                    4  1987-03-03    9.50000000000000e+001
mike                    8  1985-02-15    9.20000000000000e+001
demi               NULL        NULL       8.50000000000000e+001
finly              NULL        NULL       8.20000000000000e+001
linda                   1  1982-01-01    8.80000000000000e+001
kitty              NULL     1980-02-27    9.70000000000000e+001
jim                     8     NULL       9.80000000000000e+001
8 rows selected
```

Modifying the data type of the dynamic column named **score**:

```
dmSQL> ALTER TABLE student MODIFY DYNAMIC COLUMN score TYPE TO INT;
```

Creating an index on the dynamic column named **desk_id**:

```
dmSQL> CREATE INDEX idx1 ON student(desk_id);
```

Dropping description information of the dynamic column named **birthday**:

```
dmSQL> ALTER TABLE student DROP DYNAMIC COLUMN birthday;
```

LONG VARBINARY (BLOB)

The BLOB data type is a variable-length data type that can contain any binary value. The maximum length of BLOB columns is 8 TB. Unlike the BINARY data type, which uses zero-value bytes for padding, only the bytes entered are stored in the database.

You can enter character data by enclosing the data in single quotes (' '), the same as when entering CHAR data. However, in BLOB columns the data is stored as hexadecimal values representing the ASCII code of the characters, not as the actual characters entered.

Alternately, enter hexadecimal values directly by enclosing the data in single quotes and appending the 'x' character (' 'x) to indicate a string containing a hexadecimal value. Two digits represent all possible values for each byte in hexadecimal; use an even number of digits when entering values.

➲ **Example 1**

```
'AaBbCcDdEe'x
```

➲ **Example 2**

```
'41614262436344644565'x
```

LONG VARCHAR (CLOB)

The CLOB data type is a variable-length data type that can contain any character that can be entered from the keyboard. The maximum length of CLOB columns is 8 TB.

Unlike the CHAR data type, which uses spaces for padding, only the characters entered are stored in the database. When entering data in a CLOB column, enclose it

in single quotes (' '). Double-byte characters occupy two bytes each, account for this when specifying the length of the column.

➲ **Example 1**

```
'This is a varchar string.'
```

➲ **Example 2**

```
'This is another varchar string.'
```

NCHAR (size)

The NCHAR data type is a fixed-length data type that can contain any Unicode character. Each Unicode character occupies two bytes of storage in UTF16 Little-Endian (LE) encoding. The (size) parameter determines the number of 2 byte characters in the column. The (size) parameter must be entered when creating an NCHAR column, and may range from a minimum of 1 to a maximum of 1996.

If NCHAR data is entered into a column that is shorter than the column length, the data will be padded with spaces. When entering NCHAR data, enclose the Unicode character with single quotes and prefix the quotes with 'N'.

➲ **Example 1**

The following demonstrates the syntax of a Unicode data entry:

```
N'Unicode Data'
```

If NCHAR data is input in hexadecimal format, enclose the hexadecimal string with quotes and append a 'u' character.

➲ **Example 2**

The following demonstrates the syntax of a three-character hexadecimal Unicode data entry:

```
'610a620b63f1'u
```

When a character string is input to a Unicode column but is not prefixed by 'N', then it will automatically be converted from local code to Unicode. If Unicode characters are entered into a regular CHAR type column, then the Unicode character will be converted to the local code defined by the dmconfig.ini parameter **DB_LCode**. Characters that are not defined in the local code will be represented by □.

Synonyms for the NCHAR data type include NATIONAL CHAR(size), and NATIONAL CHARACTER(size).

NVARCHAR (size)

The NVARCHAR data type is a variable-length data type that can contain any Unicode character. Each Unicode character occupies two bytes of storage in UTF16 Little-Endian (LE) encoding. The (size) parameter determines the number of 2 byte characters in the column. The (size) parameter must be entered when creating an NVARCHAR column, and may range from a minimum of 1 to a maximum of 1996.

If NVARCHAR data is entered into a column that is shorter than the column length, the data is not padded with spaces. When entering NVARCHAR data, enclose the Unicode character with single quotes and prefix the quotes with 'N'.

➲ **Example 1**

The following demonstrates the syntax of a Unicode data entry:

```
N'Unicode Data'
```

If NVARCHAR data is input in hexadecimal format, enclose the hexadecimal string with quotes and append a '**u**' character.

➲ **Example 2**

The following demonstrates the syntax of a three-character hexadecimal Unicode data entry:

```
'610a620b63f1'u
```

When a character string is input to a Unicode column but is not prefixed by '**N**', then it will automatically be converted from local code to Unicode. If Unicode characters are entered into a regular VARCHAR type column, then the Unicode character will be converted to the local code defined by the dmconfig.ini parameter **DB_LCode**. Characters that are not defined in the local code will be represented by □.

Synonyms for the NVARCHAR data type include NATIONAL CHAR VARYING(size), NCHAR VARYING(size), NATIONAL VARCHAR(size), and NATIONAL CHARACTER VARYING(size).

OID

The OID (object identifier) data type is a special data type that provides a unique ID for each object, record or BLOB, stored in a database. A structured data type has a precision of 10 and a scale of 0, and occupies 8 bytes of storage. DBMaker automatically generates and inserts an OID with each record. The OID is internally managed and maintained by DBMaker and cannot be used directly.

The value generated for an OID is related to the storage location of objects in the database. This means that two OIDs generated consecutively may not necessarily be sequential.

The OID values act as a hidden pseudo-column in tables, and will not appear in queries such as SELECT * FROM CUSTOMERS. Explicitly select the OID column by using 'OID' as a column name in a query.

Although it is possible to use an OID in a query to select data from a table and then use the OIDs to update the table data, this is not common practice when using the SQL language. OIDs are usually used in the internal programming interface, and not directly in the interactive dmSQL environments.

REAL

The REAL data type is an approximate signed numeric data type having a mantissa with a precision of 7. Precision refers to the total number of digits to the left and to the right of the decimal point. The REAL data type uses 4 bytes of storage and has a valid input range from 3.402823466E38 to –3.402823466E38. The smallest error range are 1.175494351E-38 to –1.175494351E-38. A move involving a value larger than the allowed maximum, from a data type such as DOUBLE, fails and DBMaker displays a conversion error.

➲ **Example 1**

```
3.583837E34
```

➲ **Example 2**

```
-1.873653E-21
```

SERIAL (start)

The SERIAL data type is a special data type that provides a sequence of consecutive values. DBMaker allocates an integer number for each table contained in a database and uses those numbers to generate a unique sequence for the corresponding table. DBMaker manages and maintains these integer numbers internally. The value of each integer value is automatically increased by one each time it is used.

Providing an integer value for the optional START parameter when defining a SERIAL column to specify the first value of the column SERIAL in a number sequence, or the START parameter omitted to use the default value of ***1***. Each table in a database can have only one column with the SERIAL data type.

The internal value used to generate a SERIAL number is actually an integer value; the SERIAL data type shares all of the properties of the INTEGER data type. It is an exact signed numeric data type with a precision of 10 and a scale of 0, which occupies 4 bytes of storage. The SERIAL data type also has the same range of values as the INTEGER data type, with a maximum value of 2,147,483,646 and a minimum value of –2,147,483,648.

Place a NULL, or empty value in the SERIAL column when inserting a new row to insert a sequential number into a SERIAL column. DBMaker will insert the sequential number for that table into the SERIAL column of the new record, and automatically increase the internal value by one.

If inserting a new column, and supplying an integer value for the SERIAL instead of a NULL or empty value, DBMaker will use the supplied integer value instead of the next sequential number; the internal value will not be incremented by 1. If the supplied integer value is greater than the last sequential number generated, DBMaker will reset the sequence of generated sequential numbers to start with the supplied integer value.

➲ **Example 1**

```
100, 101, 102, 103, 104, 105, 106, 107
```

➲ **Example 2**

```
100, 101, 50, 102, 103, 110, 111, 112
```

SMALLINT

The SMALLINT data type is an exact signed numeric data type with a precision of five and a scale of zero. The SMALLINT data type uses two bytes of storage and has a maximum value of 32,767 and a minimum value of -32,768.

If attempting to move a value larger than the permitted maximum value from a data type such as INTEGER or DOUBLE, DBMaker displays a conversion error and does not move the data.

➲ **Example 1**

```
4769
```

➲ **Example 2**

```
8376
```

TIME

There are two types of TIME data, TIME literal, and TIME constant. A TIME literal displays the present time, which is an ever-changing value. A TIME constant is a fixed moment in time. Both TIME data type settings are fixed-lengths, and use 4 bytes of storage. All time values are entered in twenty-four hour format by default unless the optional '**AM**' or '**PM**' values are specified.

Both TIME data types have multiple input/output formats. If the values in the database do not appear correctly or you are unable to enter perceived valid times then, check the time input/output formats for validity.

➲ **Example 1a**

```
'22:04:05'
```

➲ **Example 1b**

```
'22:04:05't
```

➲ **Example 1c**

```
TIME '22:04:05'
```

➲ **Example 2a**

```
'10:04:05 PM'
```

➲ **Example 2b**

```
10:04:05 PM't
```

➲ **Example 2c**

```
TIME 10:04:05 PM'
```

TIMESTAMP

There are two types of TIMESTAMP, TIMESTAMP literal, and TIMESTAMP constant. A TIMESTAMP literal displays the present time, which is an ever-changing value. A TIMESTAMP constant is a fixed moment in time.

Both TIMESTAMP data type settings are a fixed-length data type that contains calendar data and the time-of-day. Both TIMESTAMP data type settings use 11 bytes of storage, has a precision of 17, and a scale of 10. Valid years range from 0001 to 9999. All time values are entered in twenty-four hour format by default unless the optional '**AM**' or '**PM**' values are specified.

Both TIMESTAMP data type settings use the input and output formats for the TIME and DATE data types to display values and determine if input values are valid. If the values in the database do not appear correctly or you are unable to enter perceived valid times then, check the time input and output formats for validity.

➲ **Example 1a**

```
'1997/01/01 10:02:03'
```

➲ **Example 1b**

```
'1997/01/01 22:02:03'ts
```

➲ **Example 1c**

```
TIMESTAMP '1997/01/01 10:02:03'
```

➲ **Example 2a**

```
'01.01.1997 22:02:03'
```

➲ **Example 2b**

```
'01.01.1997 22:02:03'ts
```

➲ **Example 2c**

```
TIMESTAMP '01.01.1997 22:02:03'
```

VARCHAR (size)

The VARCHAR data type is a variable-length data type that can contain any character that can be entered from the keyboard. VARCHAR columns have a minimum length of 1 character and a maximum length of 3992 characters. Enter a value for the size parameter when creating a VARCHAR column.

Only the VARCHAR characters entered are stored in the database. When entering data in a column, use single quotes (' '). If using double-byte characters, account for two bytes for each character when specifying the length of a column.

➲ **Example 1**

```
' This is a VARCHAR string.'
```

➲ **Example 2**

```
' This is another VARCHAR string.'
```

Media Types

Large object columns may also be specified as media types to aid in media process functions such as full text search for Microsoft™ Word™ documents. The following media types are available: MsWordType, HtmlType, XmlType, MsPPTType, MsExcelType, PDFType, MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType, and PDFFileType.

Media types are domains of existing data types; MsWordType, MsPPTType, MsExcelType, PDFType, HtmlType, and XmlType are derived from LONG VARBINARY, and MsWordFileType, HtmlFileType, XmlFileType, MsPPTFileType, MsExcelFileType, and PDFFileType are derived from FILE type columns. This is important to consider if you choose to use the ALTER TABLE function to change a column from one data type to another. The characteristics of each of the media types are similar to the characteristics of the data type from which it is derived.

The features of XMLTYPE include:

- Well-formed XML checking: inserted/updated xml content must be well-formed

- XML validation: optionally specify a validation udf when creating an xmltype column and DBMaker will validate the xml content with it
- XML data is stored in the original format
- Query with XPath search: optionally specify an xpath and use extract functions to query/locate nodes in an XML data
- Update XML content specified by XPath
- Build index on XPath extract: speed up xpath queries with indexes on frequent query xpath expression
- Altering an xmltype column or other data types to the xmltype is not allowed

➲ **Example**

```
dmSQL> CREATE TABLE minutes (id INT, meeting_date DATE, doc MSWORDFILETYPE);
dmSQL> INSERT INTO minutes VALUES (1, 3/3/2003, 'c:\meeting\20030303.doc');
```

5.3 Creating a Table

Every table is defined with a name and a set of columns. There can be up to 2000 columns in each table. Each column contains a column name and a data type. A column length might be predetermined by the data type INTEGER or a precision and scale of the DECIMAL data type only, or a starting number for columns of the SERIAL data type.

To create a table, provide the table name, column definitions, and the name of the associated tablespace. By default, if a table is not associated with a tablespace it is placed in the default user tablespace.

➲ **Example 1**

To create a table named **Employee**, enter the following:

```
dmSQL> CREATE TABLE Employee (Number SERIAL,     FirstName CHAR(15),
                              LastName CHAR(20),  Manager INT,
                              Phone CHAR(15),     HireDate DATE,
                              BirthDate DATE);
```

The command string creates the table **Employee** containing seven columns: **Number**, **FirstName**, **LastName**, **Manager**, **Phone**, **BirthDate**, and **HireDate**. The **Number** column uses a SERIAL number to provide an employee numbering schema that will increment automatically every time a new employee is added. **FirstName**, **LastName**, and **Phone** use the CHAR data type. CHAR is used for the **Phone** column as well as the names, because the phone number may have brackets, dashes, or periods in it. The last two columns, **BirthDate** and **HireDate**, use the DATE data type.

➲ **Example 2**

To create the remaining tables in the **Tutorial** database, enter the following:

```
dmSQL> CREATE TABLE Regions (Number INT NOT NULL, Name CHAR(40));dmSQL> CREATE
TABLE IDTypes (Number INT NOT NULL,  Type CHAR(50),
                              Description CHAR(255));
dmSQL> CREATE TABLE Customers (Number SERIAL,    FirstName CHAR(15),
                               LastName CHAR(20), Phone CHAR(15),
                               IDType INT,        IDRegion INT,
                               IDNumber CHAR(20), Credit SMALLINT)
dmSQL> CREATE TABLE Suppliers (Number SERIAL,  Name CHAR (50),
                                Phone CHAR(15), Contact CHAR(35));
```

```
dmSQL> CREATE TABLE MovieTypes (Number INT NOT NULL, Type CHAR(30),
                          Description CHAR(255));
dmSQL> CREATE TABLE Movies (Number SERIAL,     Name CHAR(75),
                            Year CHAR(4),      Country CHAR(30),
                            Type1 INT,         Type2 INT,
                            Type3 INT,         Type4 INT,
                            Rating CHAR(10),   Length SMALLINT,
                            Color CHAR(3),     BW CHAR(3),
                            Tape CHAR(3),      Disc CHAR(3),
                            Quantity SMALLINT, Supplier INT,
                            Data DATE,        Price FLOAT,
                            Rate SMALLINT);
dmSQL> CREATE TABLE Receipts (Number SERIAL, Customer INT,
                              Employee INT);
dmSQL> CREATE TABLE LineItems (Receipt INT NOT NULL, LineItem INT NOT NULL,
                               Movie INT,            Quantity SMALLINT,
                               Amount SMALLINT);
```

In some of the newly created tables shown above, the keyword NOT NULL is used, indicating that the column must contain data when inserting a new record. Several other keywords can also be used when creating a table.

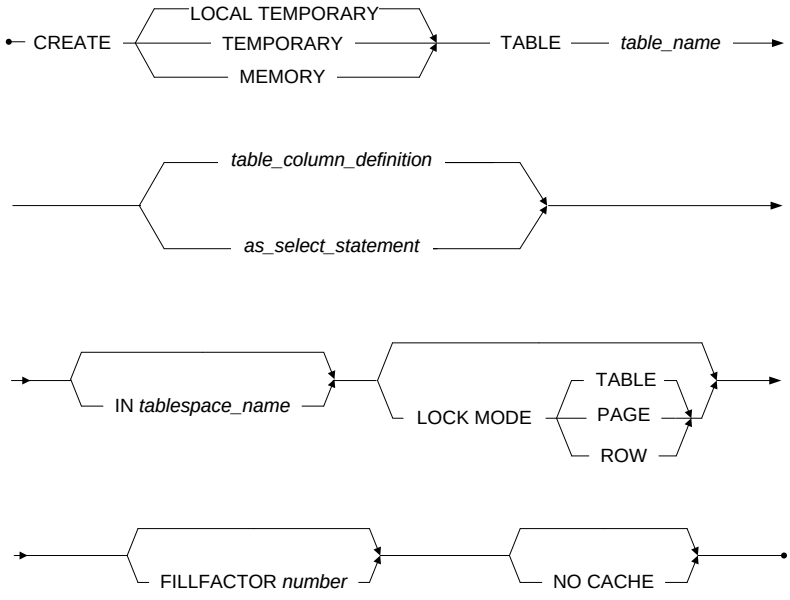


Figure 5-1: Syntax diagram for the CREATE TABLE command

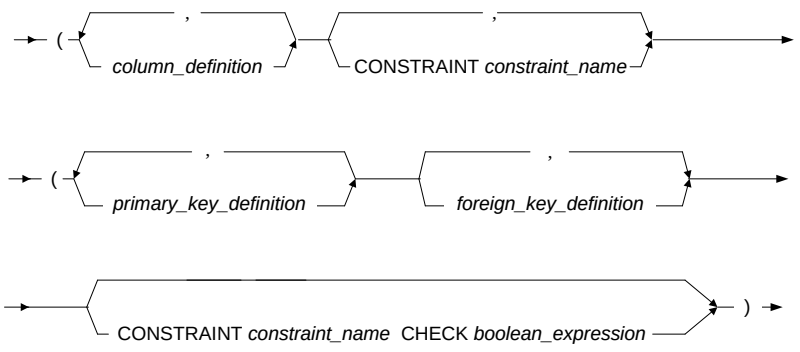


Figure 3-2 Syntax diagram for CREATE TABLE: table_column_definition

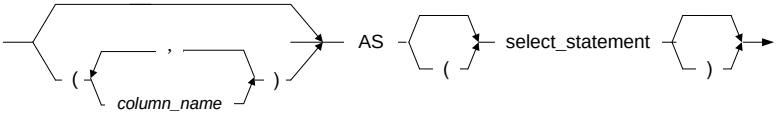


Figure 3-3 Syntax diagram for CREATE TABLE: as_select_statement

Default Values for Columns

A column can be assigned a default value. When a new row is inserted and no value is provided for a column or the column is omitted, the default value is automatically supplied. A different default value for each column in a table can be specified. If a default value is not defined for a column, the default value will be set to NULL. Acceptable default values can be constants, NULL or built-in functions. For more information about built-in functions, refer to the *SQL Command and Function Reference*.

Now DBMaker supports setting default column attribute for inserting and updating operations with the three keywords: USER, SYSTEM and ON UPDATE. The USER/SYSTEM keywords are optional. These keywords specify whether users can modify value of the column with a default value by using the INSERT/UPDATE statement. USER is used by default. The USER keyword specifies that users can modify its value, and the SYSTEM keyword specifies that users cannot modify its value. The ON UPDATE keyword also is optional. This keyword specifies that value of the column with a default value can be automatically updated when other columns' value is changed. The three keywords are mainly used in a table's definition, and for details of a table's definition, please refer to CREATE TABLE, ALTER TABLE ADD COLUMN and ALTER TABLE MODIFY COLUMN in chapter *SQL Command* of the *SQL Command and Function Reference*.

In addition, when update data, a user can use the connection option SYSTEM DEFAULT to specify whether the value of a column with SYSTEM DEFAULT attribute will be overridden to the default value. If the user sets this option to ON, the value will be updated to the default value; if the user sets this option to OFF, the original value will be updated to the value specified by the users. The default setting for this option is ***ON***. Moreover, if having assigned value to the column by using the INSERT/UPDATE statement, the user can use the connection option LOAD SYSTEM DEFAULT to specify whether the value of a column with SYSTEM DEFAULT attribute will be overridden in the process of loading the tables of database. If the user sets this option to ON, the value will be updated to the default value; if the user sets this option to OFF, the original value will be updated to the value specified by the user. The default setting for this option is ***OFF***.

➲ **Example 1**

To specify the default value of the column **nation**, in the table **tb_staff** as a constant — 'R.O.C.' and the default value of the column **joinDate** as the value of the built-in function **curdate**():

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
                                   ID INTEGER,
                                 name CHAR(30),
                             joinDate DATE DEFAULT CURDATE(),
                               height FLOAT,
                               degree VARCHAR(200),
```

```
                                picture LONG VARCHAR) IN ts_reg;
```

➲ **Example 2a**

```
dmSQL> CREATE TABLE computer(id INT, buy_time TIMESTAMP DEFAULT '2012-03-04
12:12:12', price int); //now attributes of buy_time is USER
dmSQL> INSERT INTO computer VALUES(1, '2012-10-10 10:10:20', 3400); //value of
buy_time will be replaced with '2012-10-10 10:10:20' which is specified by the
user
1 rows inserted
dmSQL> INSERT INTO computer VALUES(2, '2012-10-11 10:10:20', 5400);
1 rows inserted
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20              3400
         2 2012-10-11 10:10:20              5400
2 rows selected
dmSQL> UPDATE computer SET price=3200 WHERE id=1; //value of buy_time will not be
updated
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20              3200
         2 2012-10-11 10:10:20              5400
2 rows selected
```

➲ **Example 2b**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP DEFAULT '2012-
03-04 12:12:12' ON UPDATE); //now attributes of buy_time is USER and ON UPDATE
dmSQL> UPDATE computer SET price=3000 WHERE id=1; //value of buy_time will be
replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
========== ========================== ==========
         1 2012-03-04 12:12:12              3000
         2 2012-10-11 10:10:20              5400
2 rows selected
```

```
dmSQL> UPDATE computer SET price=3000, buy_time='2012-10-10' WHERE id=1;//value of
buy_time will be replaced with '2012-10-10' which is specified by the user
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
2 rows selected
```

➲ **Example 2c**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12'); //now attributes of buy_time is SYSTEM
dmSQL> INSERT INTO computer VALUES(3, '2012-11-10 10:10:20', 4700); //value of
buy_time will not be replaced with '2012-11-10 10:10:20' which is specified by the
user.
1 rows inserted
dmSQL> INSERT INTO computer VALUES(4, '2012-12-11 10:10:20', 2800);//value of
buy_time will not be replaced with '2012-12-11 10:10:20' which is specified by the
user.
1 rows inserted
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4700
         4 2012-03-04 12:12:12               2800
4 rows selected
dmSQL> UPDATE computer SET price=4500 WHERE id=3; //value of buy_time will not be
updated.
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4500
         4 2012-03-04 12:12:12               2800
4 rows selected
```

➲ **Example 2d**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12' ON UPDATE); //now attributes of buy_time is SYSTEM and ON
UPDATE
dmSQL> UPDATE computer SET price=4000, buy_time='2015-01-01' WHERE id=3; //value
of buy_time will be replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
    ID               BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4000
         4 2012-03-04 12:12:12               2800
4 rows selected
```

Lock Mode

The LOCK MODE option identifies the type of lock that DBMaker automatically places on objects when accessing the database. DBMaker supports three levels of lock modes table, page, and row. The page lock mode is used by default if a lock mode is not specified when a table is created.

If the lock mode is set to a higher level, such as table, the level of concurrency on database accesses will be lower, but the required lock resources and shared memory will also be smaller. If the lock mode is set to a lower level, such as row, the level of concurrency on database accesses will be higher, but the required lock resources and shared memory will be larger. In other words, if you insert or modify rows in a table with the lock mode set to table, no one else can access the table.

➲ **Example**

To specify a lock mode for a table, enter the following:

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW;
```

Fillfactor

The FILLFACTOR option is used to optimize the utilization of space on data pages by reserving space on a data page to allow for the expansion of individual records. It specifies the percentage of a page that can be filled before it will no longer permit new records to be inserted. This method ensures that the records can be accessed more efficiently by avoiding the need to retrieve information for one record from multiple pages.

➲ **Example**

To specify a FILLFACTOR of 80% for the **Suppliers** table, enter the following:

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW FILLFACTOR 80;
```

In this case, no new rows can be inserted in the data page after 80% of the page space has been filled. The value for a FILLFACTOR can be from 50% to 100 %, and the default value is 100%.

NOCACHE

The NOCACHE feature is useful when accessing large tables with a table scan. DBMaker uses page buffers in shared memory to cache retrieved data and avoid frequent disk I/O. A table scan with a larger number of data pages than the number of page buffers can exhaust all of the page buffers, causing frequent disk I/O activity. DBMaker uses one page buffer to cache the data retrieved from a table during a table scan, after the NOCACHE option has been specified when creating a table. The page buffers will not be exhausted after one large table scan.

➲ **Example**

To specify the NOCACHE option, enter the following:

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW FILLFACTOR 80 NOCACHE;
```

Temporary Tables

A temporary table for temporary data storage can be created. Temporary tables exist during a single session, and DBMaker will automatically drop the table when you disconnect from the database. Temporary tables support fast data operations and can be used only by the creator.

➲ **Example**

To create a temporary table named **test**, enter the following:

```
dmSQL> CREATE TEMPORARY TABLE test (Number SERIAL,
                                    Name CHAR(50),
                                    Phone DATE)
```

6 Data

The tables are now created, and the schema for the database is set. The database is now ready to accept data. The metaphorical filing cabinet now has files and file folders, but no information is contained in them.

In this chapter you will learn:

- How to insert data into the database by adding records, or rows
- How to change, or update data in existing records
- How to query a table to find out what data it contains
- How to remove unnecessary data from a table

6.1 Inserting

There are two ways to insert data into a table using SQL. The first method uses the standard SQL syntax, and the other uses host variables. Host variables set up an insert statement; where the data to be inserted is not known at the time the command executes. DBMaker will enter into a *value state*, allowing values for multiple records to be entered.

➲ **Example 1**

To use the SQL INSERT command, enter the following:

```
dmSQL> INSERT INTO Employee VALUES (10000, 'Gabriel', 'Davis', 10000, '228-6932',
'1/21/57', '4/24/95');
1 row inserted
```

Where **Employee** is the table name. The statement inserts the values **10000**, **Gabriel**, **Davis**, **10000**, **228-6932**, **1/21/57**, and **4/24/95** into the columns **Number**, **FirstName**, **LastName**, **Manager**, **Phone**, **BirthDate** and **HireDate**.

➲ **Example 2**

To use the SQL INSERT command for specified columns, enter the following:

```
dmSQL> INSERT INTO Employee (FirstName, LastName, Manager) values ('Greg',
'Carter', 10002);
1 row inserted
```

This statement inserts the values **Greg**, **Carter**, and **10002** into the **FirstName**, **LastName** and **Manager** columns respectively. Unspecified column values are **NULL**.

➲ **Example 3**

To use the SQL INSERT command with NULL, enter the following:

```
dmSQL> INSERT INTO Employee VALUES (NULL, 'Dean', 'Cougar');
1 row inserted
```

This statement inserts the next SERIAL value in the serial number sequence, and the values **Dean** and **Cougar** into the **FirstName** and **LastName** columns. Columns were not specified, so the values are automatically entered into the first three columns. Several other keywords may also be used with the INSERT command.

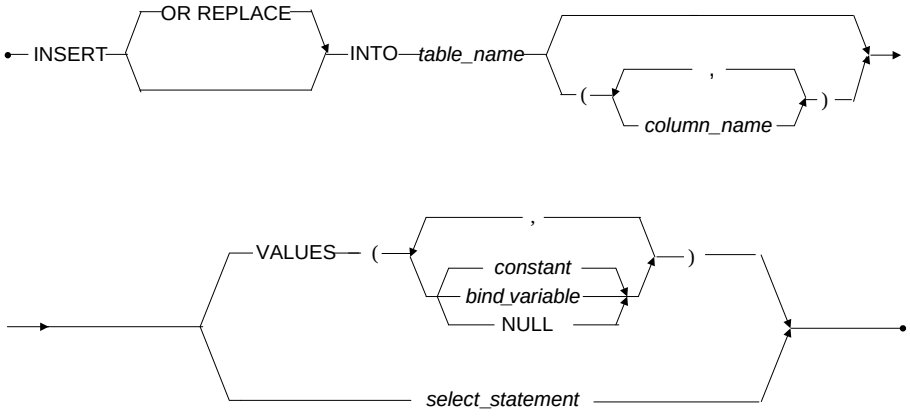


Figure 6-1: Syntax for the INSERT statement

OR REPLACE: The OR REPLACE option is optional, this option specify whether DBMaker should replace the old row with the new row if the two rows have the same value for some columns. That is to say, if the row that users will insert into a table already exists in the table (judged with the primary key or the unique index), the old row will be deleted from the table and then the new row will be inserted into the table, if not, the new row will be inserted directly into the table.

For more details of the INSERT command, please refer to chapter *SQL Command* in the *SQL Command and Function Reference*.

Inserting Using Host Variables

The syntax of INSERT with the host variables is the same as the SQL standard. An INSERT statement with host variables makes dmSQL enter into the value state with the screen prompt of dmSQL/Val>.

➲ **Example 1**

To use host variables, enter the following:

```
dmSQL> INSERT INTO Employee VALUES (?,?,?,?,?,?,?);
dmSQL/Val> NULL, 'Benson', 'Armstrong', 10002, '918-3517', '12/9/70', '3/2/93';
1 row inserted
```

```
dmSQL/Val> NULL, 'Lyn', 'Belger', 10000, '363-4511', '5/9/59', '12/6/91';
1 row inserted
dmSQL/Val> end;
```

➲ **Example 2**

To use the equivalent SQL insert command, enter the following:

```
dmSQL> insert into Employee values (NULL, 'Benson', 'Armstrong', 10002, '918-
       3517', '12/9/70', '3/2/93');
dmSQL> insert into Employee values (NULL, 'Lyn', 'Belger', 10000, '363-4511',
       '5/9/59', '12/6/91');
```

➲ **Example 3**

To use insert and host variables for values, enter the following:

```
dmSQL> insert into Employee values (NULL, ?, ?, 10002, ?, ?, ?);
dmSQL/Val> 'Murphy', 'Flaherty', '575-8846', '10/17/77', '11/17/90';
1 row inserted
dmSQL/Val> 'Taylor', 'Galbreath', '648-6633', '2/9/75', '10/22/94';
1 row inserted
dmSQL/Val> end;
```

➲ **Example 4**

To use the equivalent SQL insert command, enter the following:

```
dmSQL> insert into Employee values (NULL, 'Murphy', 'Flaherty', 10002, '575-8846',
'10/17/77', '11/17/90');
dmSQL> insert into Employee values (NULL, 'Taylor', 'Galbreath', 10002, '648-
6633', '2/9/75', '10/22/94');
```

Different Data Types

DBMaker requires the following input data types.

SMALLINT and **INTEGER**:

```
123, -252783
```

FLOAT and **DOUBLE**:

```
float:   30000.05, -234.56
double:  234.56e-257, 6.04E+23
```

CHAR and **VARCHAR**:

```
'Hello', 'DBMaker is a powerful database !'
```

BINARY:

dmSQL has two formats to identify binary types. One type is the hex format and the other is the same as the CHAR type.

Example, the binary data '61'x is the ASCII value of 'a', for the hex format:

```
'0001020304'x, '3f2eff5c'x
```

In CHAR format, every character represents one byte. However, the value stored in the database is an ASCII value of the character.

CHAR:

```
'abcedf', '!@#$%'
```

In hex format, every two hex codes represent one byte. Use hex codes 0-9, a-f (or A–F) to represent binary data. Binary data in hex format must be enclosed by single quotes and followed by a character x or X.

DATE and **TIME**:

```
'1994-12-20'd, '14:10:20't
```

DECIMAL:

```
 12.34, –0.123
```

Inserting Blob Data

DBMaker supports BLOB data. These data types are LONG VARCHAR and LONG VARBINARY. The difference between LONG VARCHAR and LONG VARBINARY is the same as the difference between CHAR and BINARY. In dmSQL, there are several ways to insert BLOB data.

➲ **Example 1**

To insert BLOB data with a SQL command, enter the following:

```
dmSQL> CREATE TABLE blob_table VALUES (lcharcol long varchar,
    2> lbincol long varbinary);
dmSQL> INSERT INTO blob_table ('this is blob data', '2d2d2d2d'x);
1 row inserted
```

➲ **Example 2**

To insert BLOB data with host variables, enter the following:

```
dmSQL> INSERT INTO blob_table VALUES (?,'5f5f5f5f5f'x);
```

➲ **Example 3**

To bind BLOB data to host variables, enter the following:

```
dmSQL/Val> 'blob using host variable';
1 row inserted
```

Alternatively, you can insert BLOB data from an external file.

➲ **Example 4**

To insert BLOB data from an external file, type the file_name in INSERT mode:

```
dmSQL/Val> &comment.txt;
1 row inserted
```

NOTE *comment.txt is a file in the current directory.*

6.2 Updating

After data has been inserted into the database, it may be necessary to change some of the values. The SQL UPDATE command is used for this purpose. DBMaker provides three ways to update column data: standard SQL, host variables or OIDs. Figure 6-2 shows the syntax for the UPDATE statement.

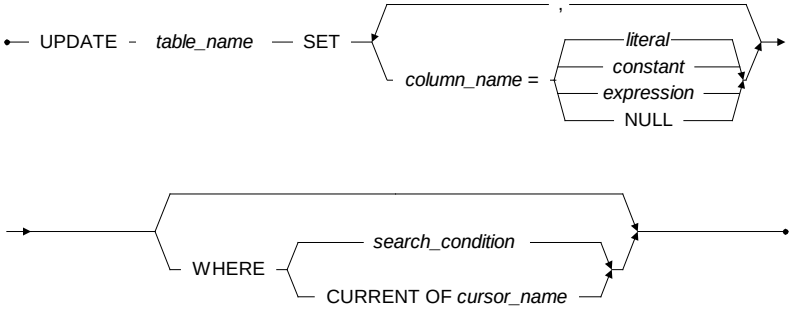


Figure 6-2: Syntax for the UPDATE statement

Updating Using Standard SQL

➲ **Example**

To update the manager file using standard SQL, enter the following:

```
dmSQL> update Employee set Manager = 10000 where LastName = 'Carter';
1 row updated
```

Where **Employee** is the table name. This statement changes the manager to **Greg Carter**.

Updating Using Host Variables

An update statement with host variables makes dmSQL enter into the value state with the dmSQL/Val> prompt. In the value state, you can enter data for the corresponding host variables. Use the END command to exit the value state and complete the update statement.

➲ **Example 1**

To update the **Employee** table using host variables, enter the following:

```
dmSQL> update Employee set Phone = ? where FirstName = ? and LastName = ?;
dmSQL/Val> '736-8376', 'Gabriel', 'Davis';
1 row updated
dmSQL/Val> '837-7847', 'Lyn', 'Belger';
1 row updated
dmSQL/Val> end;
```

➲ **Example 2**

To update the employee file using standard SQL, enter the following:

```
dmSQL> update Employee set Phone = '736-8376' where FirstName = 'Gabriel' and
LastName = 'Davis';
dmSQL> update Employee set Phone = '837-7847' where FirstName = 'Lyn' and LastName
= 'Belger';
```

The preceding entries update the **Phone** column entries on the **Employee** table for employees Gabriel Davis and Lyn Belger.

Updating Using OIDs

The OID (object id) is a special binary data type that contains the record ID of an object. Each row in a table has a unique OID. You can select an OID from a table, and then update its data. OIDs are used in the internal programming interface, and not directly in the interactive dmSQL environment.

➲ **Example**

To use an OID to update the **Employee** table, enter the following:

```
dmSQL> select oid from Employee where FirstName = 'Dean' and LastName = 'Cougar';
      oid
================
0200000002000200

1 rows selected
dmSQL> update Employee set BirthDate = '12/8/70' where oid='0200000002000200'x;
```

The entry updates the employee file for Dean Cougar's birth date.

6.3 Result Sets

The output of the SELECT statement is called a *result set*. The result set contains all data that meets the conditions specified in the SELECT command.

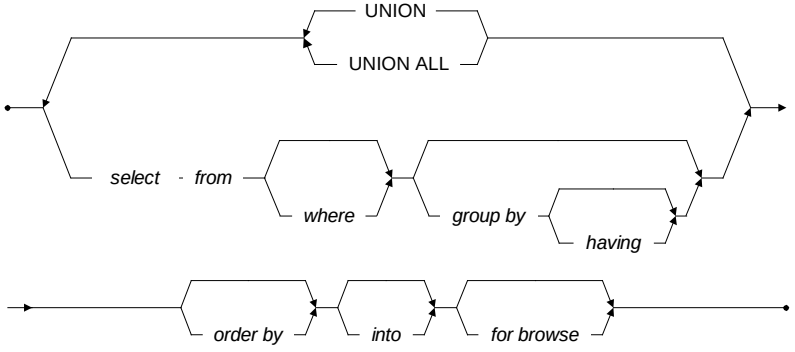


Figure 6-3: Syntax for the SELECT statement

Selecting Tables

The simplest SELECT statement can select all information in a table.

➲ **Example 1**

To select all information in a table, enter the following:

```
select * from <table_name>
```

The command selects data from all the columns in the specified table. The asterisk ('*') is used to represent all columns in a table.

Use the command shown below to select all data from the **Employee** table.

➲ **Example 2**

To select all data for employees in the **Employee** table, enter the following:

```
dmSQL> SELECT * from Employee;
```

The following is an example of returned data:

```
dmSQL> SELECT * from Employee;
```

```
dmSQL> SELECT * from Employee;

 Number   FirstName   LastName        Manager     Phone    BirthDa* HireDate
======== =========== =============== ======== =========== ======== ========
   10000 Gabriel     Davis           10000    736-8376    1957-01* 1995-04*
   10001 Greg        Carter          10000    NULL        NULL     NULL
   10002 Dean        Cougar          NULL     NULL        1970-12* NULL
   10003 Benson      Armstrong       10002    918-3517    1970-12* 1993-03*
   10004 Lyn         Belger          10000    837-7847    2059-05* 1991-12*
   10005 Murphy      Flaherty        10002    575-8846    1977-10* 1990-11*
   10006 Taylor      Galbreath       10002    648-6633    1975-02* 1994-10*

7 rows selected
```

dmSQL will display the name of the columns, the data in each of those columns and the number of rows displayed.

This provides a convenient method of finding column names in a table, when you do not remember them, instead of having to deal with system tables. It also provides a faster way to view data if you do want to see all of the columns.

You can see from the sample output that not all of the columns are fully displayed. The width of the dmSQL display line is set to ***80*** by default and will be used throughout the tutorial for formatting reasons.

➲ **Example 3**

To turn off the LINEWIDTH feature and to see all column data, enter the following:

```
dmSQL> SET LINEWIDTH off;
```

Use all of the column names in the SELECT statement to view all data in every column. In the **Employee** table, the column names are: **Number**, **FirstName**, **LastName**, **Manager**, **Phone**, **BirthDate**, and **HireDate**.

➲ **Example 4**

To view all data in every column, enter the following:

```
dmSQL> SELECT Number, FirstName, LastName, Manager, Phone, BirthDate, HireDate
from Employee;
```

The following is an example of returned data:

```
SELECT * from Employee;
  Number       FirstName          LastName          Manager        Phone
========== =============== ================== ========== ===============
```

```
 BirthDate   HireDate
=========== ===========
      10000 Gabriel         Davis                     10000 736-8376
1957-01-21  1995-04-24
      10001 Greg            Carter                    10000 NULL
NULL        NULL
      10002 Dean            Cougar                    NULL  NULL
1970-12-08  NULL
      10003 Benson          Armstrong                 10002 918-3517
1970-12-09  1993-03-02
      10004 Lyn             Belger                    10000 837-7847
2059-05-09  1991-12-06
      10005 Murphy          Flaherty                  10002 575-8846
1977-10-17  1990-11-17
      10006 Taylor          Galbreath                 10002 648-6633
1975-02-09  1994-10-22

7 rows selected
```

NOTE *Names in a database are case sensitive by default.*

If a spelling mistake occurs or the wrong case is used when entering the name of a column, an error will be returned.

➲ **Example 5**

```
dmSQL> select numbe, FirstName, LastName, Manager, Phone, HireDate, BirthDate from
Employee;
ERROR (6523): invalid column name: NUMBE
```

Only the first error encountered will be returned. After correcting an error, another error will be returned if more than one has been made until all errors have been corrected.

If a table name is wrong, an error will be returned.

➲ **Example 6**

```
dmSQL> SELECT Number, FirstName, LastName, Manager, Phone, HireDate, BirthDate
FROM employee;
ERROR (6521): table or view does not exist: SYSADM.employee
```

Selecting Columns

You can also select specific columns from a table by specifying the columns to be viewed.

➲ **Example 1**

To specify columns, enter the following:

```
dmSQL> SELECT <column_name>, <column_name>, ... FROM <table_name>
```

The command selects data in the columns specified in the table.

➲ **Example 2**

To view a list of Employees select the FirstName, LastName, and Phone columns:

```
dmSQL> SELECT FirstName, LastName, Phone FROM Employee;
```

The following is an example of returned data:

```
dmSQL>  SELECT FirstName, LastName, Phone FROM Employee;

   FirstName          LastName            Phone
============== =================== ==============
Gabriel         Davis               736-8376
Greg            Carter              NULL
Dean            Cougar              NULL
Benson          Armstrong           918-3517
Lyn             Belger              837-7847
Murphy          Flaherty            575-8846
Taylor          Galbreath           648-6633

7 rows selected
```

Selecting Rows

SQL allows us to select specific records in a database as well as selecting specific columns. This is done with the WHERE clause.

Data must meet the conditional expression defined in a WHERE clause in order to be included in the result set. Data not meeting the conditions will be excluded. Section 6.4, *Operator Types* describes the use of the WHERE clause in more detail.

➲ **Example**

```
dmSQL> SELECT * FROM Employee where <expression>;
```

6.4 Operator Types

There are three types of operators used in the expression of a WHERE clause, arithmetic operators, comparison operators, and logical operators. Each operator is used for different purposes. The comparison operators are the most frequently used.

Comparison Operators

Comparison operators are used to compare the values of two operators, and are generally used to determine whether a row should be included in a result set.

OPERATOR	DESCRIPTION
=	Specifies the Equal to mathematical sign
>	Specifies the Greater than mathematical sign
<	Specifies the Less than mathematical sign
>=	Specifies the Greater than or equal to mathematical sign
<=	Specifies the Less than or equal to mathematical sign
<>	Specifies the equivalent of a Not equal to mathematical sign
BETWEEN	Specifies a range of values
LIKE	Specifies a matching pattern
IN	Specifies a record in a database
IS	Specifies a test for special values

Figure 6-4 Comparison Operators

First, we will try a very simple conditional clause with a comparison operator to demonstrate how this works.

➲ **Example 1**

To use a comparison operator, enter the following:

```
dmSQL> SELECT * FROM Employee WHERE Number = 10006;
```

The following is an example of returned data:

```
Number   FirstName     LastName      Manager     Phone    BirthDa* HireDate
======== =========== ============== ======== =========== ======== ========
   10006 Taylor      Galbreath          10002 648-6633    1975-02* 1994-10*

1 rows selected
```

The query will select all columns from the **Employee** table by using the asterisk ('*') in the column list. The WHERE clause specifies that only employee record numbers = 10006 will be returned in the result set. In this case, records of Taylor Galbreath whose employee number is 10006 is returned. This type of query is useful when you know a single piece of data from a record and want to display the remaining data.

➲ **Example 2**

To retrieve all Employee names beginning with A, B, and C, enter the following:

```
dmSQL> SELECT * FROM Employee WHERE LastName BETWEEN 'Aa' and 'Cz';
```

The following is an example of returned data:

```
Number   FirstName     LastName      Manager     Phone    BirthDa* HireDate
======== =========== ============== ======== =========== ======== ========
   10001 Greg        Carter           10000    NULL       NULL     NULL
   10002 Dean        Cougar           NULL     NULL       1970-12* NULL
   10003 Benson      Armstrong        10002    918-3517   1970-12* 1993-03*
   10004 Lyn         Belger           10000    837-7847   2059-05* 1991-12*

4 rows selected
```

This query will select all columns from the **Employee** table by using the asterisk (') in the column list. The BETWEEN keyword used in the WHERE clause specifies that only records of an employee having a name that starts with the letters A, B, or C are returned in the result set. It does this using the BETWEEN comparison operator, which takes two arguments separated by AND. Note that although AND is used in the same way as the logical operator AND shown below, it is not a logical operator but part of the BETWEEN keyword. In order to get all names that start with: A, B, or C; the query uses the values 'Aa' and 'Cz'. If you only use the values 'A' and 'C', you will only get names that begin with A and B, and will not retrieve names that begin with the letter C. By using 'Cz' you get all the names between Aa and Cz. Someone with the names A or Czar, will not be included because they are outside the range.

➲ **Example 3**

To retrieve last names of employees that begin with 'C', enter the following:

```
dmSQL> SELECT * FROM Employee WHERE LastName LIKE 'C%';
```

The following is an example of returned data:

```
Number   FirstName     LastName     Manager     Phone    BirthDa* HireDate
======== =========== ============== ======== ========== ======== ========
   10001 Greg        Carter           10000     NULL     NULL       NULL
   10002 Dean        Cougar           NULL      NULL     1970-12*   NULL

2 rows selected
```

This query will select all columns from the **Employee** table by using the asterisk ('*') in the column list. The LIKE keyword used in the WHERE clause specifies that only records with an employee name starting with the letters "C" be returned. The wild card character '%' is used to indicate that any sort of character may follow the letter C in the string. If this wildcard is omitted, the SELECT statement will only return employees with the last name of 'C'.

Logical Operators

Logical operators are used to connect two expressions in a WHERE clause to show that there is some relationship between them.

OPERATOR	DESCRIPTION
AND	Specifies that both expressions must be true.
OR	Specifies that either expression must be true.
NOT	Specifies that an expression be excluded from an equation.

Figure 6-5 Logical Operators

➲ **Example**

To retrieve the names of movies made in Canada in 1995, enter the following:

```
dmSQL> SELECT LastName FROM Employee WHERE HireDate > 01/01/1995 AND Manager =
10000;
```

The following is an example of returned data:

```
     LastName
==================
Davis
Belger

2 rows selected
```

Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations. They are usually part of a comparison operation, and the result of calculation expressions is known until after some calculation is performed.

Operator	Description
+	Specifies a mathematical addition.
-	Specifies a mathematical subtraction/unary negation.
*	Specifies a mathematical multiplication.
/	Specifies a mathematical division.

Figure 6-6 Arithmetic Operators

6.5 Deleting

To delete rows from a table, dmSQL provides three methods standard SQL, host variables, or OIDs. Figure 6-2 shows the syntax for the UPDATE statement.

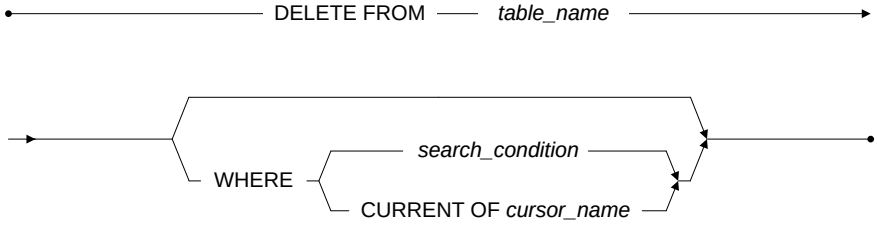


Figure 6-7: Syntax for the DELETE statement

Deleting Using Standard SQL

The following command deletes all rows in the **Employee** table.

➲ **Example 1**

To delete all rows from the **Employee** table, enter the following:

```
dmSQL> DELETE FROM Employee;
```

The following command deletes all rows meeting the condition **Number > 10030** from the **Employee** table.

➲ **Example 2**

To delete rows from the **Employee** table, enter the following:

```
dmSQL> DELETE FROM Employee where Number > 10030;
```

Deleting Using Host Variables

A delete statement with host variables makes dmSQL enter into the value state with the dmSQL/Val> prompt. In the value state, you can enter data for the corresponding host variables. Use END to exit the value state and complete the update statement.

➲ **Example 1**

To use host variables to delete employees from the Employee table:

```
dmSQL> DELETE FROM Employee WHERE FirstName = ?;
dmSQL/Val> 'Benson';
dmQL/Val> 'Murphy';
dmSQL/Val> END;
```

➲ **Example 2**

To use standard SQL with conditions for the preceding statements:

```
dmSQL> DELETE FROM Employee WHERE FirstName = 'Benson';
dmSQL> DELETE FROM Employee WHERE FirstName = 'Murphy';
```

Deleting Using OIDs

Although it is possible to manipulate data using OIDs, OID is a special binary data type used internally by the database. It is unusual to use OIDs directly in dmSQL.

➲ **Example**

To use an OID to delete employee **Taylor** from the **Employee** table:

```
dmSQL> SELECT OID FROM Employee WHERE FirstName = 'Taylor';

      OID
================
0100000002000800

1 rows selected

dmSQL> DELETE FROM Employee WHERE OID='0100000002000800'x;
```

7 Database Objects

A database may be divided into logical structures called objects. Tables, tablespaces, views, synonyms, and indexes are examples of database objects. Views, synonyms, and indexes provide convenient methods of customizing and speeding up access to data. Views and synonyms are supported to allow user-defined views and names for database objects. Indexes provide a much faster method of retrieving data from a table when querying a column with an index.

7.1 Views

DBMaker provides the ability to define a *virtual table*, called a *view*. Based on existing tables and stored in the database as a definition and a user-defined view name. The view definition is stored persistently in the database, but the actual data that you will see, is not physically stored anywhere. Rather, the data is stored in the base tables, where the viewed rows reside. A view is defined by a query, which references one or more tables or other views.

Views are a very helpful mechanism for using a database. For example, you can define complex queries once and use them repeatedly without having to re-invent them. In addition, views can be used to enhance the security of a database by restricting access to a predetermined set of rows and/or columns in a table.

Since views are derived from querying tables, views can only be queried. Users cannot update, insert, or delete from views.

Creating Views

Each view is defined by a name together with a query that references tables or other views. You can specify column names for a view that is different from them in the original table. If you do not specify any new column names, the view will use the column names from the underlying tables.

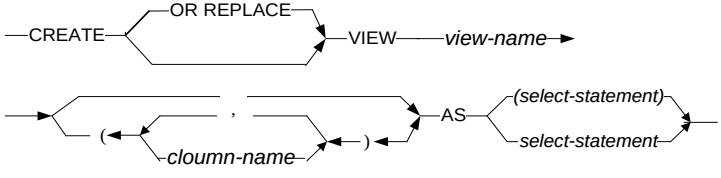


Figure 7-1 Syntax for the CREATE VIEW statement

➲ **Example 1**

To create a view on three columns in the **Employee** table, enter the following:

```
dmSQL> CREATE VIEW empView (FirstName, LastName, Telephone) AS
              SELECT FirstName, LastName, Phone FROM Employee;
```

Users can then only view the **FirstName**, **LastName**, and **Telephone** columns of the **Employee** table through **empView**.

NOTE *The query that defines a view cannot contain the UNION operator.*

Use CREATE OR REPLACE VIEW syntax. For example, a view named **vi_staff** already exists, it will allow other users to see only two columns , (**name** and **ID***)*, from the **tb_staff** table, but we need to change the view definition to allow the same users to see only three columns from the **tb_staff** table, but not change the privileges on the view. Replace the view with the SQL command shown below. Users can view three columns, (name, ID and age), from the **tb_staff** table through the view **vi_staff**.

➲ **Example 2**

To view the three columns: name, ID and age, users can recreate the view vi_staff with the following statement:

```
dmSQL> CREATE OR REPLACE VIEW vi staff (empName, empId, empAge) AS
              SELECT name, ID, age FROM tb_staff;
```

Dropping Views

You can drop a view when it is no longer required. When you drop a view, only the definition stored in the system catalog is removed with no effect on the base tables.

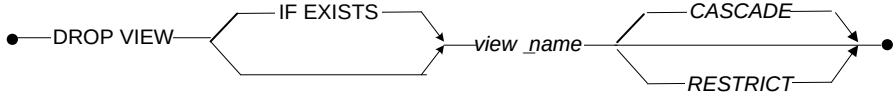


Figure 7-2 Syntax for the DROP VIEW statement

IF EXISTS: It will ensure that no error is thrown if the view does not exist.

For more details of the DROP VIEW command, please refer to chapter *SQL Command* in the *SQL Command and Function Reference*.

➲ **Example 1**

To drop a view, enter the following:

```
dmSQL> DROP VIEW empView;
```

The **empView** is removed from the **Employee** table.

➲ **Example 2**

To drop the view **vi_staff** use the DROP VIEW IF EXISTS command:

```
dmSQL> DROP VIEW IF EXISTS vi_staff;
```

7.2 Synonyms

A *synonym* is an alias, or alternate name, for any table or view. Since a synonym is simply an alias, it requires no storage other than its definition in the system catalog.

Synonyms are useful for simplifying a fully qualified table or view name. DBMaker normally identifies tables and views with fully qualified names that are composites of the owner and object names. By using a synonym, anyone can access a table or view through the corresponding synonym without having to use the fully qualified name. All synonyms in a database must be unique for DBMaker to identify them.

Creating Synonyms

If the owner of the table **Employee** is the **SYSADM**, this command creates an alias-named **Employee** for the table **SYSADM.Employee**. All database users can directly reference the table **SYSADM.Employee** through the synonym **Employee**.

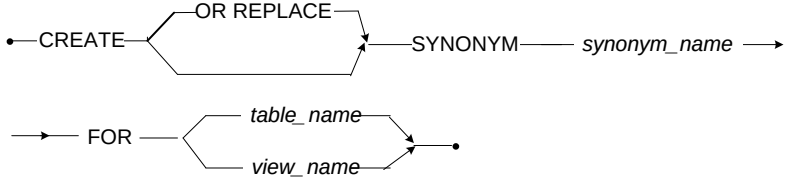


Figure 7-3 Syntax for the CREATE *SYNONYM statement*

➲ **Example 1**

Use **CREATE SYNONYM** command:

```
dmSQL> CREATE SYNONYM staff FOR SYSADM.tb_staff;
```

Assume that the owner of the table **tb_staff** is **SYSADM**. This command creates the alias **staff** for the table **SYSAMD.tb_staff**. All database users can directly reference the table **SYSAMD.tb_staff** through the synonym **staff**.

➲ **Example 2**

Use **CREATE OR REPLACE** command:

```
dmSQL> CREATE OR REPLACE SYNONYM staff FOR SYSADM.tb_staff;
```

Assume that an alias **staff** for the table **SYSAMD.tb_staff** is already exists, you can replace it without drop it.

Dropping Synonyms

You can drop a synonym that is no longer required. When you drop a synonym, only its definition is removed from the system catalog.

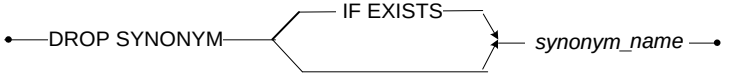


Figure 7-4 Syntax for the DROP *SYNONYM statement*

IF EXISTS: It will ensure that no error is thrown if the view does not exist.

For more details of the DROP SYNONYM command, please refer to chapter *SQL Command* in the *SQL Command and Function Reference*.

➲ **Example 1**

To drop the **staff** synonym with the DROP SYNONYM command::

```
dmSQL> DROP SYNONYM staff;
```

➲ **Example 2**

To drop the **staff** synonym with the DROP SYNONYM IF EXISTS command:

```
dmSQL> DROP SYNONYM IF EXISTS staff;
```

7.3 Indexes

An index provides support for fast random access to a row. You can build indexes on a table to speed up searching. For example, when you execute the query: "SELECT Name FROM Employee WHERE Number = 10005", it is possible to retrieve the data in a much shorter time if there is an index created on the **Number** column.

An index can be composed of more than one column, up to a maximum of 32. All the columns in a table can be used in an index.

An index can be *unique* or *non-unique*. In a unique index, no more than one row can have the same key value, with the exception that any number of rows may have NULL values. If you create a unique index on a non-empty table, DBMaker will check whether all existing keys are distinct or not. If there are duplicate keys, DBMaker will return an error message. After creating a unique index on a table, you can insert a row in the table and DBMaker will certify that no existing row already has the same key.

When creating an index, you can specify the sort order of each index column as ascending or descending. For example, suppose there are five keys in a table with the values: 1, 3, 9, 2 and 6. In ascending order, the sequence of keys in the index is: 1, 2, 3, 6 and 9, and in descending order the sequence of keys in the index is: 9, 6, 3, 2 and 1.

When you implement a query, the index order will occasionally affect the order of the data output.

➲ **Example**

If you have a table named **friends** with **name** and **age** columns, and have built an index in descending order on the **age** column, then executing the following query:

```
dmSQL> SELECT name, age FROM friends WHERE AGE > 20
```

➲ **Result**

```
      name                    age
----------------          ----------------
Jeff                                  49
Kevin                                 40
Jerry                                 38
```

```
Hughes                          30
Cathy                           22
```

When creating an index specify the *fillfactor* for it. The fillfactor denotes how dense the keys will be in the index pages. The legal fill factor values are in the range from **1%** to **100%**, and the default is ***100%***. If you update data often after creating the index, you can set a loose fill factor in the index, for example **60%**. If you never update the data in this table, you can leave the fill factor at the default value.

It is far more efficient to create an index after loading a large amount of data than to create the index before loading is finished. Before creating indexes on a table, it is recommended that you load all data first, especially if that table holds a large amount of data. If you create an index before loading the data into a table, the indexes update each time you load a new row.

Creating Indexes

To create an index on a table, specify the index name and index columns. Specify the sort order of each column as ascending (ASC) or descending (DESC). The default sort order is ascending.

In DBMaker, user can move a table to another tablespace, if the index and the table in the same tablespace, then the index will be moved to another tablespace. If the index and the table in different tablespaces, then the index will not be moved to another tablespace， so we can rebuild index in another tablespace.

Indexes can be created not only on simple columns, but also on Expression columns or User Defined Function (UDF) columns. To improve XML query performance, user can create special XML index on XML columns. A well-designed Filtered Index also can improve query performance as well as reduce index maintenance and storage costs compared with full-table indexes.

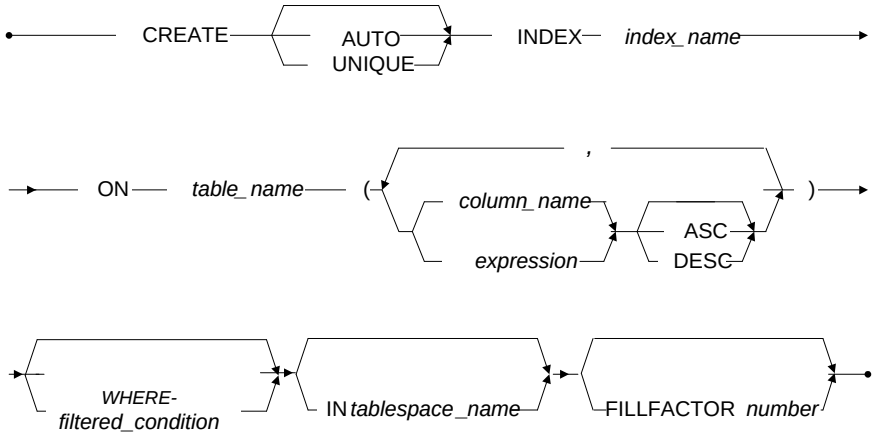


Figure 7-5: Syntax for the CREATE INDEX statement

➲ **Example 1**

To create a descending index, idx1, on the **Number** column in the **Employee** table, enter the following:

```
dmSQL> CREATE INDEX idx1 ON Employee (Number desc);
```

In addition, to create a unique index you have to specify it. Otherwise, DBMaker creates non-unique indexes.

➲ **Example 2**

To create the unique index, **idx1**, on the **Number** column in the **Employee** table, enter the following:

```
dmSQL> CREATE UNIQUE INDEX idx1 ON Employee (Number);
```

➲ **Example 3**

To create an index with a specified fill factor, enter the following:

```
dmSQL> CREATE INDEX idx2 ON Employee(Number, LastName DESC) FILLFACTOR 60;
```

➲ **Example 4**

To create an index, **idx_substr**, on the **UDF** substring (**nation,1,3**) for table **tb_salary**:

```
dmSQL> CREATE INDEX idx_substr ON tb_salary (substring(nation,1,3) desc);
```

➲ **Example 5**

To create an index use the **extract** XML UDF:

```
dmSQL> CREATE INDEX idx_extr ON tb_extract (extract(id,
'/order/items/item/@product', NULL));
```

➲ **Example 6**

To create a filtered index, **filidx_income**, using the **where** clause for table **tb_salary**:

```
dmSQL> CREATE INDEX filidx_income ON tb_salary(basepay+bonus,tax)where id>30;
```

Dropping Indexes

You can drop indexes using the DROP INDEX statement. In general, you might need to drop an index if it becomes fragmented, which reduces its efficiency. Rebuilding the index will create a denser index that is not fragmented. If the index is a primary key and referenced by other tables, you cannot drop it.

➲ **Example**

To drop the index **idx1** from the **Employee** table, enter the following:

```
dmSQL> DROP INDEX idx1 FROM Employee;
```

8 Users and Privileges

Protecting information is an important part of any database. Users should not have free access to all areas of the database, and they should not be capable of changing data at will. DBMaker provides several means of protecting data and managing access.

8.1 Security Management

Security management is important for restricting data access to only those users whom require it.

Security management provided with DBMaker:

- **User accounts** — uses user ID's and passwords to control access to the database.
- **Authority levels** — controls actions that can be performed by different users.
- **Table privileges** — lets users share some data while keeping other data private.
- **Nested groups** — simplifies granting privileges by organizing users into groups.

A unique user ID and an optional password identify each user able to access the database. All users have an authority level for their user accounts. These user authority levels specify the type of access permitted. There are five authority levels in DBMaker: CONNECT, RESOURCE, DBA (Database Administrator), SYSDBA and SYSADM (System Administrator). There may be any number of CONNECT, RESOURCE, DBA, or SYSDBA user accounts, but there is only one SYSADM account. The SYSADM account is reserved for the person who sets up and maintains the database and user accounts.

Table privileges control access to data. By using table privileges, users can access data created in the tables. Use table permissions to allow users to perform operations on the table data which determines if they can view, insert, delete, and update data, create an index, reference a table, or alter a table by adding new columns. Table privileges can also be granted to everyone; by using the PUBLIC keyword, the privileges on a table are granted to all users.

8.2 Authority Levels

To connect to a database, a user must be granted authority. A user with *connect* authority has very limited access to the database. They cannot create any new objects, such as tables or views. They can only view or alter data in tables that have privileges granted to the PUBLIC table. They cannot alter or view any tables created by other users until the owner of the table or a user with DBA authority or higher has granted table privileges to them. After table privileges have been granted, they may use any table privileges that have been granted to them or to the PUBLIC table.

Resource

Resource authority allows a user to create new tables in the database and drop any tables they previously created. As the owner of a table, they may also create and drop views and indices on tables they created. They may also grant and revoke table privileges on any tables they own. They cannot alter or view any tables created by other users until the owner of the table or a DBA has granted privileges to them or to the PUBLIC table.

DBA

DBA authority allows a much greater degree of control over the database. It gives a user all privileges on all tables and views in the database. A DBA can create new tables or view, alter, and grant, table permissions on all tables. A DBA can also create new database structures, for instance tablespaces and data files. To control access for a number of users at a time, a DBA can also create and remove groups and add and remove users from those groups. However, a DBA user cannot change the authority level of a current user, grant new users permission to connect to the database, or change user passwords.

SYSDBA

Users with SYSDBA authority can both grant or revoke CONNECT, RESOURCE and DBA to other users, change other users' passwords, except users with SYSADM or SYSDBA authority, and set ACL (Access Control List) for users with lower authority. Users with SYSDBA have all the privileges of DBA authority and only users with SYSADM authority can grant or revoke SYSDBA from users. For users with SYSDBA authority, if the SYSADM revokes SYSDBA, the users still has DBA authority, but if the SYSADM revokes DBA, the users has neither SYSDBA nor DBA authority.

SYSADM

There is only one SYSADM for each database. The SYSADM ID is assigned by default to the user who created the database. The SYSADM has complete control over the database and can add new tables, views, alter and drop the tables of any user, grant table permissions for any tables, create new tablespaces, or data files. Only the SYSADM and a SYSDBA can grant, revoke, or change user authority levels, and change a user's password.

8.3 New Users

Only users with SYSADM or SYSDBA authority can grant CONNECT authority and add new users to a database.

➲ **To add new users to the database:**

1. Start the database and log in as **SYSADM**
```
dmSQL> START DB tutorial SYSADM <password>;
```

2. Connect to **Tutorial** with the **SYSADM** password
```
dmSQL> CONNECT TO tutorial SYSADM <password>;
```

3. Choose one of the following to create a new user account without a password
```
dmSQL> GRANT CONNECT TO Judy;
dmSQL> GRANT CONNECT TO Judy NULL;
dmSQL> GRANT CONNECT TO Judy "";
```

4. Alternatively, create a new user account with a password
```
dmSQL> GRANT CONNECT TO Judy secret;
```

User Access

When you have connected to the database as the SYSADM, you are ready to add a new user. Add new users using the GRANT keyword. The GRANT command specifies usernames, passwords, and their authority levels.

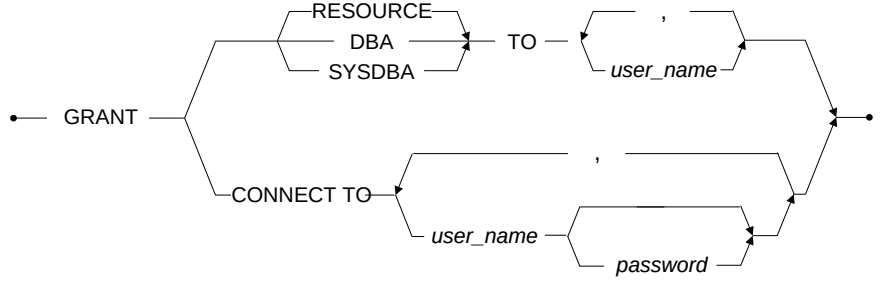


Figure 8-1: Syntax of the GRANT statement

The user ID may contain a maximum of 128 alphanumeric characters. If the first character of the user ID is a number, enclose the entire user ID in double quotes ("").

The user ID's case sensitivity is dependent on the database configuration parameter **DB_IDCap**: if **DB_IDCap** = 0 all identifiers (table names, usernames, and passwords) are case-sensitive. It is possible to have two different user accounts with the names **judy** and **Judy**. The default setting of **DB_IDCap** is ***1***, so unless this parameter is specified all identifiers will be case-insensitive.

Like the user ID, the password may also contain a maximum of sixteen alphanumeric characters. If the first character is a number, enclose the password in double quotes.

Grant all new users the CONNECT authority before promoting them a higher authority level (RESOURCE, DBA, SYSDBA and SYSADM). When creating a new user account, choose to assign a password or create the account with no password and let the user create one later. For security reasons, assign a temporary password when you create the account, ask the user to change it when they first log on.

Every time a user logs on to the database, they must type the username exactly as it appears. For consistency, it is best to use the same format when you create new user accounts.

Multiple Users

Create new user accounts for multiple users at the same time. The syntax is the same as adding one user, apart from specifying multiple usernames and passwords on the command line.

➲ **To add two new users without assigning passwords:**

1. Start the database and log in as SYSADM
```
dmSQL> START DB tutorial SYSADM <password>;
```

2. Connect to **Tutorial** with the SYSADM password
```
dmSQL> CONNECT TO connect to tutorial SYSADM <password>;
```

3. Choose one of the following to create new user accounts without a password
```
dmSQL> GRANT CONNECT TO Tom, Judy;
dmSQL> GRANT CONNECT TO Tom "", Judy "";
dmSQL> GRANT CONNECT TO Tom NULL, Judy NULL;
```

4. Alternatively, create new user accounts with passwords
```
dmSQL> GRANT CONNECT TO Tom secret1, Judy secret2;
```

8.4 Promoting Authority Level

The GRANT keyword is also used to promote an existing user's authority level. The procedure is the same as granting CONNECT authority to a new user. A user may already have a password for their user account, do not specify a new password unless a new person is going to use the account.

➲ **Example**

To promote a user from CONNECT authority to RESOURCE or DBA authority:

```
dmSQL> GRANT RESOURCE TO Judy;
dmSQL> GRANT DBA TO Judy;
```

Granting DBA authority to a user does not automatically give them RESOURCE authority. This becomes important when demoting a user authority level. If you want to give a user both RESOURCE and DBA authority, use the grant command twice, once to grant the RESOURCE authority level, and again to grant the DBA authority level.

Multiple Users

You can also promote multiple users at the same time. The syntax is the same as for promoting one user, but specify multiple usernames on the command line. The users must be promoted to the same authority level. It is not possible to promote multiple users to different authority levels with one command.

➲ **Example**

To promote two new users:

```
dmSQL> GRANT RESOURCE TO Tom, Judy;
dmSQL> GRANT DBA TO Tom, Judy;
```

8.5 Demoting Authority Level

Demoting a user authority level is similar to promoting user authority level; use the REVOKE keyword instead.

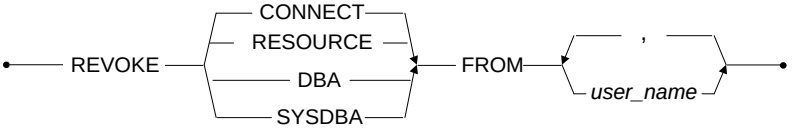


Figure 8-2: Syntax of the REVOKE statement

When revoking an authority level from a user, that user drops to the next lower authority level granted to them. For example, a new user granted CONNECT authority and then granted DBA authority, without RESOURCE authority. After revoking DBA authority from the user, they will only have CONNECT authority.

➲ **Example 1**

To demote a user from DBA to the next lower authority level:

```
dmSQL> REVOKE DBA FROM Judy;
```

➲ **Example 2**

To demote a user from RESOURCE to CONNECT authority:

```
dmSQL> REVOKE RESOURCE FROM Judy;
```

NOTE *If a user was granted RESOURCE authority and then DBA, revoke both DBA and RESOURCE authority to demote the user to CONNECT authority.*

➲ **Example 3**

To demote a user from DBA authority to CONNECT authority:

```
dmSQL> REVOKE DBA FROM Judy;
dmSQL> REVOKE RESOURCE FROM Judy;
```

8.6 Removing Users

Use the REVOKE keyword to remove an existing user. The procedure for removing an existing user is the same as for demoting a user's authority level. Once a user has had CONNECT authority revoked, that user is removed from the list of people who can connect to the database.

➲ **Example**

To remove a user from a database:

```
dmSQL> REVOKE CONNECT FROM Judy;
```

8.7 Passwords

The SYSADM assigns a temporary password to a new user when creating their user account. If the user wants to change this password or if the SYSADM did not assign a temporary password, the user can change the password or set a new password by using the ALTER PASSWORD command. The SYSADM can also reassign a new password to a user. Use this method when a user forgets a password.

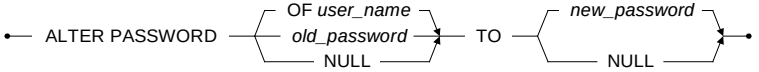


Figure 8-3: Syntax for the ALTER PASSWORD statement

➲ **Example 1**

To create a password a user would enter:

```
dmSQL> ALTER PASSWORD NULL TO newpass;
dmSQL> ALTER PASSWORD "" TO newpass;
```

➲ **Example 2**

To change a temporary password a user would enter:

```
dmSQL> ALTER PASSWORD X9elx4 TO newpass;
```

A user can also remove a password by changing their current password to NULL.

➲ **Example 3**

To change a password to NULL, enter:

```
dmSQL> ALTER PASSWORD oldpass TO NULL;
dmSQL> ALTER PASSWORD oldpass TO "";
```

The SYSADM can change or remove any user password. No other users can change someone else's user password. To change a user password, the SYSADM does not have to know the current password.

➲ **Example 4**

To change a user password, a SYSADM would choose one of the following:

```
dmSQL> ALTER PASSWORD OF Judy TO newpass;
dmSQL> ALTER PASSWORD OF Judy TO NULL;
dmSQL> ALTER PASSWORD OF Judy TO "";
```

8.8 Managing Groups

When a database becomes very large and has many users, it becomes increasingly difficult to grant user privileges on an individual basis. To solve this problem, DBMaker defines users in groups. Using groups, you can collect all users whom require the same database privileges to simplify authorization management. Table privileges can be granted or revoked for all users in a group at the same time and rights to a number of users at the same time, simplifying authorization management.

Creating

To create a new group, use the CREATE GROUP command.

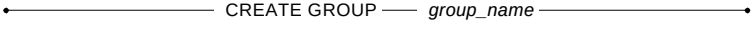


Figure 8-4: Syntax for the CREATE GROUP statement

The group name may contain alphanumeric characters. The group name can be any length, but only the first eight characters are used. Be careful when naming groups, both of the groups **CompanyEmployee** and **CompanyExecutives** would have the same significant characters **CompanyE**. Trying to create the second group with the same significant characters will result in an error.

The group name is also case sensitive, so the names **companyExecutives** and **CompanyExecutives** create two different groups, **companyE** and **CompanyE**. To avoid these situations, limit group names to eight or less characters and avoid long descriptive names.

➲ **Example**

To create a group for all marketing personnel, use:

```
dmSQL> CREATE GROUP marketing;
```

Add all marketing department personnel to the new group called marketing. Then grant privileges to the group and all members will have access to the same objects.

Adding Members

Add one or several users a time.

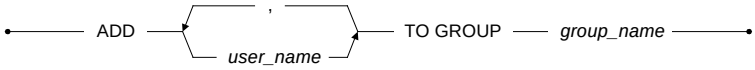


Figure 8-5: Syntax for the ADD TO GROUP statement

➲ **Example 1**

To add one user, **Judy**, to the group **SalesRep**:

```
dmSQL> ADD Judy TO GROUP SalesRep;
```

➲ **Example 2**

To add the users **Judy**, **Jeff**, and **Trent** to the **SalesRep** group:

```
dmSQL> ADD Judy, Jeff, Trent TO GROUP SalesRep;
```

Removing Members

Use the REMOVE GROUP command to remove a single user or to remove multiple users.

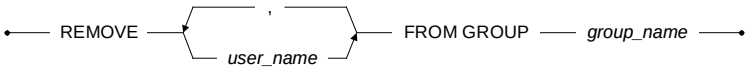


Figure 8-6: Syntax for the REMOVE FROM GROUP statement

➲ **Example 1**

To remove one user, **Judy**, from the **SalesRep** group:

```
dmSQL> REMOVE Judy FROM GROUP SalesRep;
```

➲ **Example 2**

To remove the users **Judy**, **Jeff**, and **Trent** from the **SalesRep** group:

```
dmSQL> REMOVE Judy, Jeff, Trent FROM GROUP SalesRep;
```

Dropping

If a group of users is empty or is no longer required, drop the group by using the DROP GROUP command.

Figure 8-7: Syntax for the DROP GROUP statement

If the group is not yet empty, remove all users with the REMOVE FROM GROUP command above.

➲ **Example**

To drop the **SalesRep** group, ensure that it is empty and enter:

```
dmSQL> DROP GROUP SalesRep;
```

Nested Groups

Nested groups further enhance the functionality of groups. To create a nested group, use a group name as a user ID and use the ADD TO GROUP command. To remove the nested group, use the REMOVE FROM GROUP command.

➲ **To create a group named NYSales:**

1. Type the following command
   ```
   dmSQL> CREATE GROUP NYSales;
   ```
2. Add it to the **SalesRep** group
   ```
   dmSQL> ADD NYSales TO GROUP SalesRep;
   ```
3. Then remove it
   ```
   dmSQL> REMOVE NYSales FROM GROUP SalesRep;
   ```

8.9 Table Level Privileges

There are seven table privileges that the owner of a table or users with DBA authority or higher can use. Four of the privileges, SELECT, INSERT, DELETE, and UPDATE, allow users to view and alter the data in a table. The three remaining privileges permit users to create indexes-INDEX, change the table definition-ALTER, and place referential integrity constraints on the table-REFERENCE. Table privileges can also be used to set changes to specified columns.

Select

The SELECT privilege permits users to view any data in a table or view. The table owner or users with DBA authority or higher may grant the SELECT privilege.

Insert

INSERT privilege permits users to insert new data in a table. Grant INSERT privilege for an entire table or for specified columns by including a column name list after the INSERT keyword. The table owner or users with DBA authority or higher may grant the INSERT privilege.

Delete

The DELETE privilege permits a user to delete data from a table. The table owner or users with DBA authority or higher may grant the DELETE privilege.

Update

UPDATE privilege permits users to update any of the values in a table. You can grant UPDATE privilege for an entire table or for specified columns by including a column name list after the UPDATE keyword. The table owner or users with or higher authority may grant the UPDATE privilege.

Index

The INDEX privilege permits users to create an index for a table. The table owner or users with DBA authority or higher may grant the INDEX privilege.

Alter

The ALTER privilege permits users to alter the table definition for a table. The table owner or users with DBA authority or higher may grant the ALTER privilege.

Reference

The REFERENCE privilege permits users to create foreign keys for a table. Grant the REFERENCE privilege for an entire table or for specified columns. To grant the REFERENCE privilege on specified columns, include a column name list after the REFERENCE keyword. The table owner or users with DBA authority or higher may grant the REFERENCE privilege.

8.10 GRANT Privileges

The owner of the table or users with DBA authority or higher may grant table privileges to any user. Grant the privileges for an entire table or for specified columns.

Use the GRANT command to assign privileges for tables.

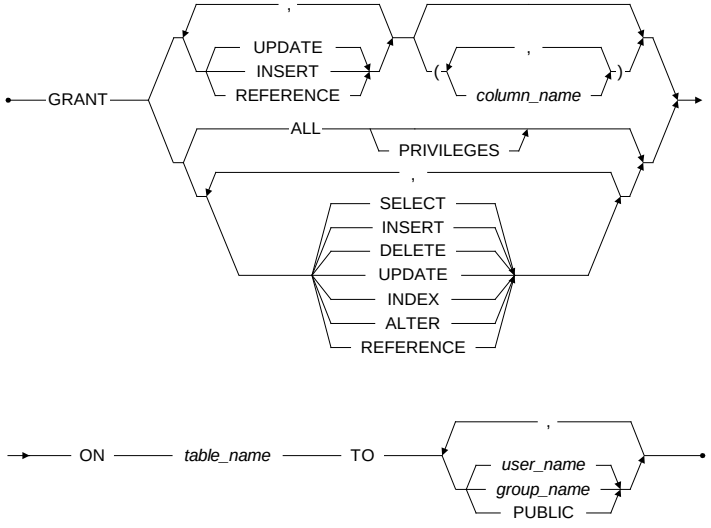


Figure 8-8: Syntax for the GRANT statement

Privileges cannot be granted on a table and specified columns at the same time. Use two commands to grant privileges on the entire table and privileges on specified columns. Privileges can be granted to a single user, groups, or to all users by using the PUBLIC keyword.

GRANT Table Privileges

➲ **Example 1**

To grant the SELECT privilege on the **SalesRep** table to **Judy**, enter the following:

```
dmSQL> GRANT SELECT ON SalesRep TO Judy;
```

It is also possible to grant more than one privilege at the same time. List the privileges on the command line, separated with a comma.

➲ **Example 2**

To grant the SELECT and UPDATE privileges on the **SalesRep** table to **Judy,** enter the following:

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Judy;
```

Grant all table privileges to a user by listing all of the keywords on the command line, or use the ALL keyword provided by DBMaker.

➲ **Example 3**

To grant the entire table list of privileges: SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, AND REFERENCE on the **SalesRep** table to **Judy**:

```
dmSQL> GRANT ALL ON SalesRep TO Judy;
```

Grant privileges to multiple users by specifying multiple usernames, separated by commas.

➲ **Example 4**

To grant the SELECT and UPDATE privilege on the **SalesRep** table to **Judy** and **Jeff**, enter the following:

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Judy, Jeff;
```

Grant privileges to a group of users or multiple groups by specifying group names, separated by commas.

➲ **Example 5**

To grant the SELECT and UPDATE privileges on the **SalesRep** table to the groups **Personnel** and **SalesMgr**, enter the following:

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Personnel, SalesMgr;
```

NOTE *It is not possible to grant privileges on multiple tables at the same time.*

GRANT Column Privileges

It is possible to grant the INSERT, UPDATE, and REFERENCE privileges only on specified columns. When granting privileges on columns, it is not possible to grant other privileges on the entire table in the same command.

➲ **Example 1**

To grant the INSERT privilege on the **Name** column of the **SalesRep** table to **Judy**, enter the following:

```
dmSQL> GRANT INSERT (Name) ON SalesRep TO Judy;
```

➲ **Example 2**

To grant the INSERT privilege on more than one column to **Judy**, list the columns separated by commas:

```
dmSQL> GRANT INSERT (Name, Age, RepOffice, Title) ON SalesRep TO Judy;
```

When granting multiple privileges with one command, all of the privileges must act on the same columns. DBMaker does not support granting privileges on different columns in a single command.

➲ **Example 3**

To grant **Judy** the UPDATE, INSERT and REFERENCE privileges on the **Name** and **Age** columns of the **SalesRep** table , enter:

```
dmSQL> GRANT INSERT, UPDATE, REFERENCE (Name, Age) ON SalesRep TO Judy;
```

Grant privileges to multiple users and groups for columns the same way as granting privileges on tables; list the user or group names separated by commas.

8.11 REVOKE Privileges

The table owner or users with DBA authority or higher can revoke privileges from any user on a table or from specified columns. Use the REVOKE command to remove privileges.

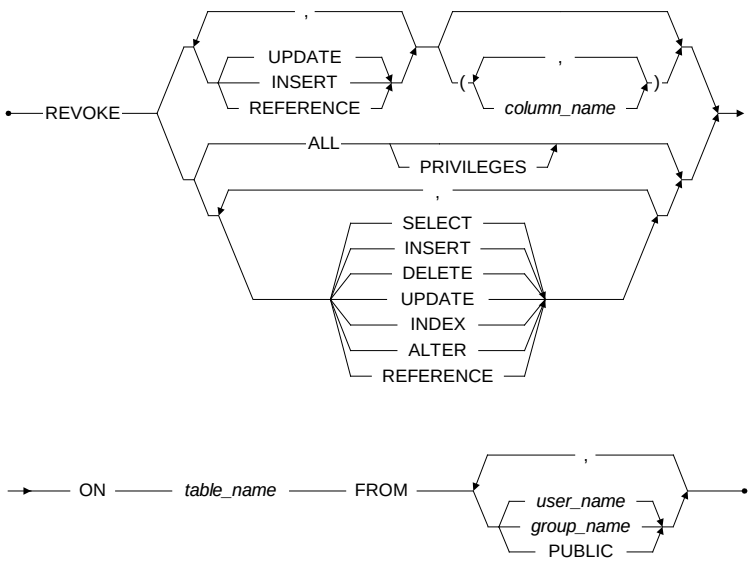


Figure 8-9: Syntax for the REVOKE statement

Privileges on a table and specified columns may not be revoked at the same time. Use two commands to revoke privileges on the entire table and privileges on specified columns. Use the PUBLIC keyword to revoke privileges for a single user, group, or all users.

REVOKE Table Privileges

➲ **Example 1**

To revoke from **Judy** the SELECT privilege on the **SalesRep** table, enter:

```
dmSQL> REVOKE SELECT ON SalesRep TO Judy;
```

Revoke more than one privilege by listing the privileges to be revoked on the command line, separated with a comma.

➲ **Example 2**

To revoke from **Judy** the SELECT and UPDATE privileges on the **SalesRep** table, enter:

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Judy;
```

Revoke all table privileges, (SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX and REFERENCE), from a user by listing all of the keywords, or use the ALL keyword provided by DBMaker.

➲ **Example 3**

To revoke from **Judy** the entire list of table privileges on the **SalesRep** table, enter:

```
dmSQL> REVOKE ALL ON SalesRep TO Judy;
```

Revoke privileges from multiple users by specifying multiple usernames, separated by commas.

➲ **Example 4**

To revoke from **Judy** and **Jeff** the SELECT and UPDATE privileges from the **SalesRep** table, enter:

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Judy, Jeff;
```

Revoke privileges from a group of users or multiple groups by replacing the usernames with the group names.

➲ **Example 5**

To revoke from the **Personnel** and **SalesMgr** groups the SELECT and UPDATE privileges on the **SalesRep** table, enter:

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Personnel, SalesMgr;
```

NOTE *It is not possible to grant or revoke privileges on multiple tables at the same time.*

REVOKE Column Privileges

It is possible to revoke the INSERT, UPDATE and REFERENCE privileges on specified columns. If you are revoking privileges on columns, it is not possible to revoke other privileges on the entire table with the same command.

➲ **Example 1**

To revoke from **Judy** the INSERT privilege on the **Name** column of the **SalesRep** table, enter:

```
dmSQL> REVOKE INSERT (Name) ON SalesRep TO Judy;
```

➲ **Example 2**

To revoke from **Judy** the INSERT privilege on more than one column of the **SalesRep** table, list the columns separated by commas:

```
dmSQL> REVOKE INSERT (Name, Age, RepOffice, Title) ON SalesRep TO Judy;
```

When revoking more than one privilege with one command, all of the privileges must be in use on all of the columns.

➲ **Example 3**

To revoke from **Judy** the UPDATE, INSERT, and REFERENCE privileges on the **Name** and **Age** columns of the **SalesRep** table, enter:

```
dmSQL> REVOKE INSERT, UPDATE, REFERENCE (Name, Age) ON SalesRep TO Judy;
```

Revoke privileges on columns for multiple users and groups in the same way as revoking privileges on tables; list the user or group names separated by commas.

9 Database Recovery

In every database management system, the possibility of a hardware or software failure always exists. A RDBMS may fall victim to these types of failures without warning. After a failure occurs, a RDBMS should have some method of recovering the information. This is one of the main advantages that a RDBMS has over the file-based systems they replaced.

DBMaker incorporates advanced data protection features to prevent data loss and downtime due to failures. These features allow DBMaker to ensure the reliability of a database and the consistency of data by providing recovery, backup, and restoration features.

This chapter will introduce you to the types of failures that may occur in a database, and what steps you should take to prevent the loss of data. As in previous chapters, examples are given using the dmSQL command line tool, however, this chapter also contains examples of how to use the JServer Manager Tool to backup and restore a database.

9.1 Types of Failures

Two of the most common types of database failures are *system failures* and *media failures*. When a failure occurs, there is the possibility of data inconsistency or data loss from a database. A RDBMS should provide facilities to recover from a failure and for replacing a damaged database with a backup copy.

System

A system failure, known as an instance failure, is a failure of the main memory. System failures may be caused by a power failure, an application or operating system crash, a memory error, or some other reason. This results in the unexpected termination of a database management system.

When a system failure occurs, applications and active transactions may terminate abnormally. The exact state of a transaction in progress or a transaction not completely written to disk cannot reliably be determined after a system failure; these types of transactions require recovery. The most common method of protecting against system failures is the use of a *transaction log* or a *journal file*.

Media

Media failure, known as disk failure, is the failure of a disk storage system. Media failures are caused by physical trauma to the disk, a head crash, fire, or exposure to vibration or g-forces outside of its physical operating limits.

When a media failure occurs, there is generally nothing to prevent data loss on the affected disk. Backup and restoration facilities can still successfully restore a database.

9.2 Recovery Methods

The goals of recovery after a database failure are to ensure committed transactions are reflected in the database, ensure that uncommitted transactions are not reflected in the database, and to return to normal operation as quickly as possible while insulating users from problems caused by the failure.

DBMaker uses journal files and *checkpoints* to achieve these goals. The journal files and checkpoints work together to ensure recovery of all transactions in as short a time as possible.

Journal Files

Journal files provide a real-time and historical record of all changes made to a database and the status of each change. In the event of a system failure, the historical record of changes maintained in the journal files allows DBMaker to recover and redo changes made by transactions that completed but were not written to disk, or undo changes made by transactions that terminated abnormally.

If a database is running in backup mode, the journal files will also store additional information that DBMaker can use. This information will remain in the journal files until they backed up; after backing up the journal files DBMaker will free the space for use by new transactions. During the restoration process, DBMaker will add the information from the backup journal files to a backup copy of the database. This allows you to backup only the journal files that contain the changes made to the database between full backups.

Checkpoint Events

A *checkpoint* is a system event in which the database is brought to a clean state. DBMaker writes all journal records and all dirty data pages from its internal memory buffers to disk and reclaims journal blocks that are no longer required for backup or

recovery purposes. DBMaker can reclaim journal blocks that contain non-active transactions completed before the start of the oldest active transaction.

Startup time reduces when taking a checkpoint after an instance failure. DBMaker writes the time of the last checkpoint and a list of all active transactions at the time of the checkpoint, to the journal file header. During database recovery, DBMaker uses information to determine which transactions should be undone, redone, and ignored.

DBMaker will automatically take a checkpoint when the journal files are full to try to reclaim some journal blocks for reuse. If the checkpoint cannot reclaim enough space to complete the current transaction, the transaction will be aborted. DBMaker will also automatically take a checkpoint when the database starts and shuts down, and during an online backup.

Database administrators can initiate a checkpoint manually by executing the CHECKPOINT command. The optimal interval between two checkpoints depends on the frequency of activity in the database, the average size of transactions, and the size and number of journal files. Since these factors may vary significantly from database to database, you may only be able to determine the optimal interval through experience. Manual checkpoints reduce the amount of time required to start, terminate, and backup a database or a full journal.

Checkpoints may require a significant amount of time to complete, depending on the size and number of transactions since the last checkpoint. Any transactions that are active when a checkpoint occurs are paused while DBMaker calculates which journal records it can reclaim. When DBMaker starts to write journal records and dirty data pages to disk the transactions proceed.

Recovery Steps

DBMaker provides support for automatic recovery when you try to start a database after a system failure or when an error occurs during startup. During the recovery process, DBMaker always performs two separate steps redo and undo.

The first step in the recovery process is to redo all changes made to the database that are recorded in the journal. It is possible for a transaction to have completed before the system failure without having all of the changes made by the transaction written to

the database. However, these changes are stored in the journal and can be written to the database during this step. At the end of this step, the database contains the changes made by all committed transactions and the changes made by all uncommitted transactions.

The second step in the recovery process is to undo all of the changes made by transactions that were not completed before the system failure occurred. The exact state of a transaction in progress cannot be reliably determined in the event of a system failure, so it cannot continue to completion. These incomplete transactions must be removed since a transaction is self-contained by definition and must either complete successfully and change the data, or fail and leave the data unchanged. At the end of this step, the database contains the changes made by all committed transactions, but does not contain any changes made by uncommitted transactions.

DBMaker also provides support for starting a database after a media failure or after a system failure, which causes inconsistencies in a database that cannot be repaired during the automatic recovery process. In these cases, the database will fail to start and you would normally need to restore a database from a backup copy. However, if you have never backed up your database, you can force the database to start by setting the forced-start mode using the **DB_ForcS** keyword in the **dmconfig.ini** file. This will allow you to start the database and unload the unaffected data. For more information on the forced-start mode, refer to the *Database Administrator's Guide*.

9.3 Types of Backup

Backups are used to protect your database from media failures or other media errors. After a media failure, one or more of your database files may be physically damaged and unusable. You can use the most recent backup to replace the damaged files and reconstruct a database in a case like this.

There are three main types of database backups: *full backups, differential backups* and *incremental backups.* When you want to perform a backup, there are two different states a database may be in, *online* or *offline*. DBMaker supports several different combinations of backup types and database states.

Full Backup

A full backup creates a copy of all data and journal files, providing a copy of the entire database at one point in time. You can optionally create a backup copy of the **dmconfig.ini** file as well, preserving any custom configuration settings you may have for a database. DBMaker allows you to perform a full backup while a database is online or offline.

Full backups archive the entire database, so they require a large amount of storage space. However, you can restore a database relatively quickly using a full backup since you can simply copy the backup files over the damaged originals. A full backup to can restore a database to the point in time that the last full backup was performed and database files copied.

Differential Backups

A differential backup is based on the latest full backup of the data. This is known as the *base* of the differential or the differential base. A differential base must exist before making a differential backup.

Differential backups contain only data that has changed since the differential base was generated. Typically, a differential base is used for several successive differential

backups. During a restore, the full backup and its corresponding differential backup will produce a full database.

Differential backups include data files(all DB files and BB files）. Journal files are excluded because they change too frequently. Only useful journal blocks are copied during differential backups.

Incremental Backup

An incremental backup creates a copy of only the journal files that have changed since the last backup. These files provide a copy of the changes made to the database since the last backup. Since an incremental backup only contains changes made to the database, you must perform a full or differential backup before the incremental backup in order to restore a database. DBMaker allows you to perform an incremental backup only while a database is online.

Note that an incremental backup is composed of journal files which record all transactions since the backup mode (DB_BMode) is on. When DBMaker Database is running on normal mode (DB_SMode = 1), before doing an incremental backup, a full backup or a differential backup must has been done; when on replication mode (DB_SMode= 4) , an incremental backup can be done without a full backup or a differential backup.

Incremental backups archive only journal files, so they require a small amount of storage space. However, it may take more time than a full backup to restore a database since the RDBMS must take the time to roll over all transactions in the backup journal files. You can use an incremental backup together with a full or differential backup to restore a database to any point in time between the previous full or differential backup and when the incremental backup completed.

Offline Backup

An *offline backup* must be performed after a database has been shut down. The Database Administrator must schedule a time to shut down the database, and notify all users so they can disconnect from the database. Offline backups can be

inconvenient for users, since they must remember to complete all active transactions and disconnect from the database before it is shut down. A database must be offline during a full backup.

An RDBMS does not need to provide the capability to backup a database offline, since you can backup the database with operating system commands after it is shut down. DBMaker allows you to perform an offline backup using this method, but also provides Server Manager, an easy-to-use graphical tool that allows you to perform offline backups without resorting to using operating system commands.

Online Backup

An *online backup* can be performed while a database is running. The Database Administrator does not have to shut down the database, and users do not need to disconnect. Online backups are more convenient for users, since no action is required on their part. DBMaker can perform full, differential and incremental backups while online.

An RDBMS must provide the capability to backup a database online, since it is still running and still has users connected. DBMaker allows you to perform online backups manually using dmSQL and operating system commands, but also provides Server Manager, an easy-to-use graphical tool that allows you to perform online backups without resorting to operating system commands.

Online Incremental to Current Backups

DBMaker also supports an additional backup type called online incremental to current.

The difference between an online incremental backup and an online incremental to current backup in a database with multiple journal files is minor, but important. In an online incremental backup DBMaker will backup all journal files that have been used since the last backup, excluding the active journal file. In an online incremental to current backup DBMaker will backup all journal files that have been used, including the active journal file. This means that an online incremental backup can restore a

database up to the point in time the last committed transaction was written to the last full journal file, while an online incremental to current backup can restore a database up to the point in time the active journal file was backed up.

In a database with only a single journal file, an online incremental backup and an online incremental to current backup are the same. In this case, the only journal file is the active journal file. DBMaker will backup this single journal file in both types of incremental backup.

Backup Combinations

You can perform full, differential and incremental backups using DBMaker, but they are mutually exclusive; you cannot perform a full, a differential and an incremental backup together. Similarly, DBMaker supports backup execution while the database is offline or online. Additionally, DBMaker allows various combinations of full, differential and incremental backups with online or offline backups.

DBMaker supports the following backup combinations: offline full backups, online full backups, online differential backups and online incremental backups. DBMaker also supports an additional backup type known as *online incremental to current*.

The difference between an online incremental backup and an online incremental to current backup in a database with multiple journal files is minor, but very significant. In an online incremental backup DBMaker will backup all journal files that have been used since the last backup, excluding the active journal file. In an online incremental to current backup, DBMaker will backup all journal files that have been used including the active journal file. This means that an online incremental backup can restore a database up to the point in time the last committed transaction was written to the last full journal file, while an online incremental to current backup can restore a database up to the point in time the active journal file was backed up.

In a database with only a single journal file, an online incremental backup, and an online incremental to current backup are the same. In this case, the only journal file is the active journal file. DBMaker will backup this single journal file in both types of incremental backup.

9.4 Backup Modes

Backup mode determines whether DBMaker can perform online incremental backups, and the type of data that will be backed up during an incremental backup. It also determines when DBMaker will free journal blocks that belong to inactive transactions for use by other transactions. DBMaker has three backup modes, NONBACKUP, BACKUP-DATA, and BACKUP-DATA-AND-BLOB.

NONBACKUP Mode

NONBACKUP mode does not provide protection for data inserted or updated since the last full backup. In this mode, a database cannot perform online incremental backups. A database can use the journal to fully recover from instance failure, but a media failure may result in loss of data. Journal blocks not in use by an active transaction can be reused immediately after a checkpoint, but once they are overwritten, you can only restore the database to the point in time of the last full backup.

BACKUP-DATA Mode

BACKUP-DATA mode provides protection for data, excluding BLOB data, inserted or updated since the last full backup. In this mode, a database can perform an online incremental backup, but only non-BLOB data will be stored in the backup files. A database can use the journal to fully recover from instance failure, and can partially recover from media failure. Although you can use the last backup to restore the database to the point in time of the media failure, any changes to BLOB data will be lost. Journal blocks not in use by an active transaction can only be reused after a checkpoint has taken place and the journal file has been backed up.

BACKUP-DATA-AND-BLOB Mode

BACKUP-DATA-AND-BLOB mode provides protection for all data, including BLOB data, inserted or updated since the last full backup. In this mode, a database can perform an online incremental backup, and all data will be stored in the backup files. A database can use the journal to fully recover from instance failure, and can also fully recover from disk failure. You can use the last backup to completely restore the database to the point in time of the media failure, including all BLOB data. Journal blocks not in use by an active transaction can only be reused after a checkpoint has taken place and the journal file has been backed up.

Tablespace BLOB Backup Mode

DBMaker normally applies the backup mode setting to the entire database, applying all tablespaces in the database with the same backup mode. If the database is in BACKUP-DATA-AND-BLOB mode, DBMaker will record all changes to data, (including BLOB data), in the journal. Recording BLOB data in the journal can quickly exhaust journal space, producing frequent backups and large backup file sizes.

This may be necessary if you cannot afford to lose any BLOB data. In many cases, you may also be backing up non-critical BLOB data at the same time. Situations like this make it difficult for you to decide which backup mode to choose. To prevent this type of situation from occurring, DBMaker allows you to modify the database backup mode for individual tablespaces when you create them.

If you want to backup BLOB data in a specific tablespace, you can use the BACKUP BLOB ON option when you execute the CREATE TABLESPACE command. If you do not want to backup BLOB data in a specific tablespace, you can use the BACKUP BLOB OFF option when you execute the CREATE TABLESPACE command.

The backup mode of each tablespace will then depend on the combination of database backup mode and tablespace backup mode as follows:

- If the database is running in BACKUP-DATA-AND-BLOB mode and a tablespace was created with the BACKUP BLOB ON option, DBMaker will backup BLOB data in that tablespace.

- If the database is running in BACKUP-DATA-AND-BLOB mode and a tablespace was created with the BACKUP BLOB OFF option, DBMaker will not backup BLOB data in that tablespace.
- If the database is running in BACKUP-DATA mode, DBMaker will not backup BLOB data regardless of whether a tablespace was created with the BACKUP BLOB ON or BACKUP BLOB OFF option.

DBMaker will use ***BACKUP BLOB ON*** by default. All changes to BLOB data in a tablespace will be recorded in the journal file when the database is in BACKUP-DATA-AND-BLOB mode.

Backup Mode of File Objects

In addition to backing up regular and BLOB data in the database, users may choose to back up file objects. File objects are backed up only during automatic full backups initiated by the backup daemon. Users should first start the backup server, set the full backup schedule, and set the backup directory.

There are two types of file objects: user file objects and system file objects. The database administrator may choose to back up system file objects, system and user file objects, or neither. The **dmconfig.ini** keyword **DB_BkFoM** specifies the file object backup mode.

- **DB_BkFoM** = 0: Do not backup file objects
- **DB_BkFoM** = 1: Backup system file objects only
- **DB_BkFoM** = 2: Backup both system and user file objects

When backing up file objects (**DB_BkFoM** = 1, 2), the backup server copies all external files of file objects to the " **FO** " subdirectory under the directory specified by **DB_BkDir** keyword. The schedule follows the full backup schedule specified by **DB_FBkTm** and **DB_FBkTv**.

➲ **Example**

An excerpt from a **dmconfig.ini** file containing related keywords:

```
[MyDB]
```

```
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 01/05/01 00:00:00        ; begins from midnight at May 1, 2001.
DB_FBKTV = 1-00:00:00                ; interval is every one day.
DB_BkDir = /home/dbmaker/backup     ; backup directory
DB_BkFoM = 2                        ; backup both system and user file objects
```

Since the file object backup mode is 2, the backup server will copy all external database file objects to the "*/home/dbmaker/backup/FO*" directory. If the **FO** subdirectory does not exist, the backup server will create it.

The files in *FO* subdirectory are renamed with a sequential number. For example, if the name of the original external file is "**/DBMaker/mydb/fo/ZZ000123.bmp**", the backup server would copy it to the **FO** subdirectory and rename it '**fo0000000344.bak**', meaning it is the 344th file object. The mapping between the source full file name and its new name is recorded in the file object mapping list file, **dmFoMap.his**. For more information about the file object mapping list file, refer to section 9.7, *Backup History Files*

The backup server will also move the previous version of file objects to the **FO** subdirectory under the old backup directory specified by **DB_BkOdr**.

Database administrators should consider that enabling file object backup requires more time for a full backup. The cost of complete full backup includes (1) copying the previous full backup if **DB_BkOdr** is set; (2) copying all database files; (3) copying all journal files; and (4) copying all file objects if **DB_BkFoM** is set. Also, ensure that there is enough disk space in the backup directory specified by **DB_BkDir** for all backup files to avoid backup failure.

Backup Mode of Stored Procedures

There are three types of stored procedures, SQL stored procedures, ESQL stored procedures and JAVA stored procedures. Because source codes are written into a database, SQL stored procedures are backed up as regular data during a full backup. In addition to backing up regular, BLOB data, and file objects in the database, users may choose to back up ESQL stored procedures and JAVA stored procedures. The two types of stored procedures are backed up only during automatic full backups initiated by the backup daemon. Users should first start the Backup Server, set the full backup

schedule, and set the backup directory. For more information about settings of a full backup, please refer to section 15.6, *Backup Server* in *Database Administrator's Guide*.

Users can specify the backup mode of stored procedures by setting **DB_BkSPm**.

- **DB_BkSPm** = 0: Do not backup ESQL stored procedures and JAVA stored procedures
- **DB_BkSPm** = 1: Back up all ESQL stored procedures and JAVA stored procedures

➲ **Example**

An excerpt from a **dmconfig.ini** file containing related keywords is shown below:

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 14/05/01 00:00:00        ; begins from midnight at May 1, 2014
DB_FBKTV = 1-00:00:00               ; interval is every one day
DB_BkDir = /home/dbmaker/backup     ; backup directory
DB_BKSPM = 1                        ; backup all ESQL stored procedures and JAVA
stored proce
```

The default directory of the two types of stored procedures's backup files is the subdirectory named **SP** under the directory specified by the **DB_BkDir** keyword. If users have set the **DB_BkOdr** keyword in the **dmconfig.ini** file, in the process of doing a full backup, the previous backup sequences of the two types of stored procedures will be moved into the subdirectory also named **SP** under the old backup directory specified by **DB_BkOdr**, and then these backup sequences will be deleted from the default directory of stored procedures.

In the process of backing up the two types of stored procedures, Backup Server firstly generated a file named dmSpBk.his. Then the Backup Server will copy stored procedures' files, meanwhile, files that use filename extension .s0 and .b0 and are used to load the backed stored procedures will be created. For ESQL stored procedures, the files that need to be backed up are source files and object files; for JAVA stored procedures, the files that need to be backed up only are object files. For convenience of rebuilding stored procedures, ESQL stored procedures' source files will be renamed in the format sp_nameowner.ec. For example, if the ESQL stored procedure's name is **y1**, and its owned by SYSADM, the copied source files will be renamed **y1SYSADM.ec**.

The file **dmSpBk.his** is used to record backup information. For more information about the stored procedures backup list file, please refer to section 14.7, *Backup History Files*

Database administrators should consider that enabling file object backup requires more time for a full backup. The cost of complete full backup includes (1) copying the previous full backup if **DB_BkOdr** is set; (2) copying all database files; (3) copying all journal files; (4) copying all file objects if **DB_BkFoM** is set; (5) copying all ESQL stored procedures and JAVA stored procedures if **DB_BkSPm** is set. Also, ensure that there is enough disk space in the backup directory specified by **DB_BkDir** for all backup files to avoid backup failure.

Compressing Backup Files

Database files can become very large and a large amount of free space is required to store backup files. DBMaker now supports compressing backup files. To enable or disable this feature, you can set the keyword **DB_BkZip** in **dmconfig.ini**, or change BkZip with the system procedure **SetSystemOption** while the database is running.

DB_BkZip = 1: Compresses the backup files

DB_BkZip = 0: Backup files are not compressed (default)

The compression format is GZIP, so you can use any GZIP-compatible tool to read the compressed file.

NOTE *FO files are not compressed, even if you set DB_BkZip to enable compressing backup files.*

Setting Backup Mode

DBMaker provides several different methods to set the backup mode. The method you choose depends on whether your database is online or offline, and whether you are more comfortable editing the configuration file directly, using the dmSQL command line utility, or using the JServer Manager graphical utility.

Modifying the backup mode of a database to provide a higher level of backup protection has an effect on journal usage. The journal begins recording previously unrecorded changes to data, before modifying the backup mode. As a result, it is necessary to perform a full or differential backup when you change the backup mode. This provides a starting point for the backup journal files to update during the restoration process.

No extra steps are required when modifying the backup mode of a database to provide a lower level of backup protection. The journal simply stops recording changes to data. DBMaker will use the previous full or differential backup as a starting point for the backup journal files to update during the restoration process. However, some changes to data may be lost if you restore a database after changing to a lower level of backup protection.

You can change the backup mode of a database offline using the **dmconfig.ini** file or JServer Manager. Since the backup mode affects journal usage, you must perform an offline full backup before you start the database with the new backup mode setting. When changing the backup mode offline, you may change from any backup mode to any other backup mode without restriction, providing you make a full backup when going from a lower level of backup protection to a higher level. To learn how to use JServer Manager to change the backup mode, refer to the *JServer Manager User's Guide*.

You can change the backup mode of a database online using dmSQL. Since the backup mode will affect journal usage, you must change the backup mode from a lower level of backup protection to a higher level (i.e., from NONBACKUP to BACKUP-DATA or BACKUP-DATA-AND-BLOB mode, or from BACKUP-DATA to BACKUP-DATA-AND-BLOB mode) between the start and finish of a full backup period.

➲ **Example 1**

To BEGIN BACKUP, SET DATA BACKUP ON, END BACKUP DATAFILE, and END BACKUP JOURNAL, enter:

```
dmSQL> BEGIN BACKUP;
dmSQL> SET DATA BACKUP ON;
dmSQL> END BACKUP DATAFILE;
dmSQL> END BACKUP JOURNAL;
```

➲ **Example 2**

To BEGIN BACKUP, END BACKUP DATAFILE, SET DATA BACKUP ON, and END BACKUP JOURNAL, enter:

```
dmSQL> BEGIN BACKUP;
dmSQL> END BACKUP DATAFILE;
dmSQL> SET DATA BACKUP ON;
dmSQL> END BACKUP JOURNAL;
```

DBMaker does not allow you to go from a higher level of backup protection to a lower level unless you change to NONBACKUP mode first. If you want to change from BACKUP-DATA-AND-BLOB to BACKUP-DATA mode, you must first change to NONBACKUP mode and then follow the rules above for changing from a lower level of backup protection to a higher level. You may change the backup mode from BACKUP-DATA-AND-BLOB or BACKUP-DATA to NONBACKUP at any time; you do not need to do this between the start and finish of a full backup period. However, during runtime, users can't directly change backup mode from NONBACKUP to BACKUP-DATA-AND-BLOB mode, or from BACKUP-DATA-AND-BLOB to BACKUP-DATA mode. For more information on performing an online full backup, see *Online Full Backups* later in this chapter.

USING THE DMCONFIG.INI FILE TO SET THE BACKUP MODE

If the database is offline, you can change the backup mode directly using the **DB_BMode** keyword in the **dmconfig.ini** file. The next time you start the database, the new backup mode will be used. If the database is online, changing the value of the **DB_BMode** keyword will have no effect until the database is shut down and restarted. You must remember to perform an offline full backup if you are going from NONBACKUP to BACKUP-DATA or BACKUP-DATA-AND-BLOB mode, or from BACKUP-DATA to BACKUP-DATA-AND-BLOB mode.

➲ **To set the backup mode using the dmconfig.ini configuration file:**

1. Using any **ASCII** text editor, open the **dmconfig.ini** file on the database server.
2. To change the **backup** mode, locate the **database configuration section** for the database.
3. Change the value of the **DB_BMode** keyword to one of the following values:

```
0 - NONBACKUP mode
```

```
1 - BACKUP-DATA mode
2 - BACKUP-DATA-AND-BLOB mode
```

4. To begin using the new backup mode, restart the database.

If the **DB_BMode** keyword is not present in the database configuration section for the database you want to change the backup mode for, you will have to add it to the database configuration section. You can add the keyword on a separate line anywhere between the start of the database configuration section and the start of the next configuration section; the order the keywords appear in is not important. If you do not specify a value for **DB_BMode** the default value of ***0*** (NONBACKUP mode) will be used.

USING DMSQL TO SET THE BACKUP MODE

If the database is online and you are comfortable using the dmSQL command line utility, you can change the backup mode using the SQL SET command. You must execute this command during an online full or differential backup. The new backup mode will be enabled as soon as the command is executed.

➲ **To set the backup mode using the dmSQL command line utility:**

1. Using dmSQL, connect to a database.
2. Using the BEGIN BACKUP command, begin an online full backup.
3. Change the backup mode during the full backup period by issuing one of the following SET commands:

```
dmSQL> SET BACKUP OFF;
dmSQL> SET DATA BACKUP ON;
dmSQL> SET BLOB BACKUP ON;
```

4. Complete the **online full backup**.

The SET BACKUP OFF command corresponds to NONBACKUP mode, the SET DATA BACKUP ON corresponds to BACKUP-DATA mode, and the SET BLOB BACKUP ON command corresponds to BACKUP-DATA-AND-BLOB mode.

USING JSERVER MANAGER

If the database is offline, you can change the backup mode using the JServer Manager graphical utility. JServer Manager will automatically change the value of the

DB_BMode keyword in the **dmconfig.ini** file. The next time you start the database, the new backup mode will be used. If the database is online, changing the value of the **DB_BMode** keyword will have no effect until the database is shut down and restarted. You must remember to perform an offline full backup if you are going from NONBACKUP to BACKUP-DATA or BACKUP-DATA-AND-BLOB mode or from BACKUP-DATA to BACKUP-DATA-AND-BLOB mode. For directions on how to set the backup mode offline using the JServer Manager graphical utility refer to the *JServer Manager User's Guide*.

9.5 Offline Full Backup

To perform an offline full backup, you must have read permission on the database files from the operating system, and write permission on the backup directory from the operating system. If you have to shut down the database first, you must have DBA authority or higher.

You can perform an offline full backup regardless of the backup mode; the database may be running in NON-BACKUP, BACKUP-DATA, or BACKUP-DATA-AND-BLOB mode. Using an offline full backup, you can restore the database to the point in time the database was shut down.

Offline Full Backup using dmSQL

➲ **To perform an offline full backup using dmSQL:**

1. Notify all users that the database will be shut down at a specified time and ask them to disconnect from the database before the specified time.
2. If the database is running, shut down the database using the **TERMINATE DB** command. If there are any errors while shutting down the database, restart the database, correct the problem, and shut it down again.
3. Examine the **dmconfig.ini** file and determine which files and directories, including the file object directory, require backup.
4. Use operating system commands or utilities to copy the database files, (including data files, journal files, file objects, and the **dmconfig.ini** file), to the backup directory or backup device.

Offline Full Backup Using JServer Manager

➲ **To perform an offline full backup using JServer Manager:**

1. Start the **JServer Manager** application on the database server
2. Select **Backup Database** from the main console to see the list of different backup options

3. Select **Off Line Full Backup** from the Backup window. The **Off Line Full Backup** window appears.

4. Select a database from the **Database Name** drop-down list box. JServer Manager will prompt you to log onto the database.

5. Enter your User-ID in the **User ID** field.

   NOTE *Any user with the DBA authority or higher can back up the database.*

6. Enter a password in the **Password** field.

7. Click **OK**. A single user connection is established to the database. The **Off Line Full Backup** window is displayed with a list of operating system files to be backed up.

8. Select a new path for the backup directory by clicking the browse button (…).

9. Select **OK** to save all files in the backup directory.

   NOTE *If the files already exist in the backup directory, the database administrator may choose to overwrite them.*

10. Examine the **dmconfig.ini** file and determine the location of the *file object* directory.

11. Use *operating system* commands or *utilities* to copy all *file objects* to the **backup** directory or **backup** device.

9.6 Backup Server

Although DBMaker provides methods for backing up a database manually, you must still remember to perform the backups on a regular basis. To solve this problem, DBMaker provides a convenient and easy way to perform fully automated online full, differential and incremental backups using Backup Server. Please note that Backup Server can only perform an online backup, since only after database startup, Backup server can startup .

The database administrator may also perform backups during runtime with the JServer Manager utility 'Backup by Backup Server'

Backup Server runs in the background and performs online full, differential and incremental backups on a regular schedule, as journal files become full, or both. This flexibility is possible because Backup Server and the database server communicate to determine when a backup should occur, the type of incremental backup to perform, and which backup options to change. Backup Server starts at the same time as the database server, and continues running until you either stop it or shut down the database server. The backup server is

When performing full backups, Backup Server will copy the last full backup from the backup directory to the old directory. Then, it will copy all database files including journal files and **dmconfig.ini** to the backup directory, over writing the previous full backup.

When performing differential backups, Backup Server copies only data (all DB files and BB files) files. Journal files are excluded because the journal files change more frequently. So, when doing differential backup, only useful journal blocks are copied. Data files in read-only tablespaces are excluded for differential backup.

When performing incremental backups, Backup Server will copy necessary journal files to the backup directory.

There are several options used to configure Backup Server. These options control the filename format of the backup files, the location of the backup directory, the location of the old directory, the schedule Backup Server uses to perform backups, interval and

maximum number of differential backup to retain after a full backup, the amount a journal file must fill before Backup Server performs an incremental backup, and the way Backup Server saves backup files.

Backup Server also allows backup-related configuration settings to be made during the run time with the dmSQL **SetSystemOption** stored procedure, that is to say, BKSVR, BKDIR, BKITV, DBKTV, BKTIM, BKFUL, BKFOM, BKZIP, BKCMP, BKRTS, BKCHK, FBKTM, FBKTV, DBKMX, BKODR, BKFRM can be changed during the run time with the **SetSystemOption** system stored procedure.

Starting Backup Server

Backup Server is a daemon and its life cycle is as long as the database server. Users do not have to explicitly start Backup Server after setting the **DB_BkSvr** keyword, since DBMaker will automatically start Backup Server while starting the database. Backup Server is disabled by default. Backup server will only be started when the database is starting in muti-user mode.

Backup Server has two states: inactive and active.Users can control the state of backup server with **DB_BkSvr** When **DB_BkSvr** is set to 0, the Backup Server is inactive. Backup Server will not respond to any backup request, namely the Backup Server will not perform any backup; when **DB_BKSvr** is set to 1, the Backup Server is active. Backup Server will response to a variety of backup requests, and then users can do any backup.

To activate the backup server, there are three methods: setting the value of the **DB_BkSvr** keyword to 1 in the **dmconfig.ini** file, changing BkSvr to 1 with *call setsystemoption('bksvr','1')* after the database is started and using Run Time Setting in Jserver Manager to change the backup setting when the dataase is running.

Before doing backup with Backup Server, users need to set some parameters to specify how to do a backup. For example, backup directory, compact backup mode and so on.

The following is how to set these parameters:

- Users can set related keywords in **dmconfig.ini** before starting the database. The next time users start the database, backup server will use these keywords to initialize associated parameters.
- If the database has been started, users can use **Run Time Setting** in Jserver Manage to alter values of parameters. Additionally, users can set parameters with *call SetSystemOption('option_name','value')* to set parameters. Please note that individual parameters only can be set with set syntax, such as set backup OFF; set data backup ON; set blob backup ON.

When Backup Server is activated, and the appropriate backup parameters is set in the **dmconfig.ini** configuration file, you can call the system stored procedure **SetSystemOption** to begin a backup. The stored procedure can be used by any client tool or user application.

➲ **Example**

The syntax to do online full, differential and incremental backup is:

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1'); //do full backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','2'); //do incremential backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3'); //do differential backup
```

USING DMCONFIG.INI TO START BACKUP SERVER

If the database is offline, you can enable Backup Server directly using the **DB_BkSvr** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will also start. If the database is online, changing the value of the **DB_BkSvr** keyword by **dmconfig.ini** configuration file will have no effect until the database is shut down and restarted.

➲ **To start Backup Server using the dmconfig.ini file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to enable Backup Server.
3. Ensure the backup mode of the database is either BACKUP-DATA or BACKUP-DATA-AND-BLOB mode. The database is in BACKUP-DATA mode if the value of **DB_BMode** is set to 1, and it is in BACKUP-DATA-AND-BLOB mode if the value of **DB_BMode** is set to 2.

4. Change the value of the **DB_BkSvr** keyword to 1 to enable Backup Server.
5. Restart the database to begin using Backup Server.

USING DMSQL TO START BACKUP SERVER

When a database is online, the Backup Server can be dynamically enabled using the dmSQL command line tool as shown below.

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR','1');
```

Users can change BkSvr with *Call SetSytemOption('BkSvr', '1')*. To change BkSvr and the value of **DB_BkSvr** in the **dmconfig.ini** configuration file at the same time, users can using *Call SetSystemOptionW('option','value')*.

When Backup Server is activated, and the appropriate backup parameters is set in the **dmconfig.ini** configuration file, you can call the system stored procedure **SetSystemOption** to begin a backup. The stored procedure can be used by any client tools and user applications.

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1') ; //do full backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','2'); //do incremential backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3'); //do differential backup
```

The syntax to change incremental backup interval is:

```
dmSQL> CALL SETSYSTEMOPTION('bkitv', 'Interval')
```

USING JSERVER MANAGER TO START BACKUP SERVER

Regardless of the online status of a database, you can enable Backup Server dynamic using the JServer Manager graphical utility. JServer Manager automatically changes the value of **DB_BkSvr** keyword in the **dmconfig.ini** configuration file. If the database is offline, the next time you start the database, Backup Server will also start. For directions on starting Backup Server while offline using JServer Manager, refer to the *JServer Manager User's Guide*.

➲ **To start Backup Server while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.

4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. Select a backup mode:
7. To select data backups only, select the Backup Data Only option button.
8. To backup data and BLOB files, select the Backup Data and BLOB option button.
9. Click the **Save** button.
10. Click the Cancel button to return to the **Start Database** window

NOTE *f the **DB_BkSvr** keyword is not present in the database configuration section for the database you want to enable Backup Server for, JServer Manager will add it automatically.*

Differential Backup Filename Format

The following is differential backup filename format:

DTimeStamp_DataFileName.dif(2)

D — differential backup identification (required)

TimeStamp — number of seconds since January 1, 1970 (00:00:00 GMT)

DataFileName — name of the database the data file belongs to

.dif — differential backup file objects are appended with the file extension **.dif**. If no differential backup source file exists with the full backup, the file extension name must be **.dif2**.

Support the first differential backup is performed at 2009/12/01 14:11, then generated differential backup filenames are D1259647860_DBNAME.BB.dif, D1259647860_DBNAME.DB.dif, D1259647860_DBNAME.SBB.dif and D1259647860_DBNAME.SDB.dif, journal filename is D1259647860_DBNAME.JNL.

Incremental Backup Filename Format

Backup filename format is <I><TimeStamp><_><DB_BKFRM>, e.g., I1234567890_%2F%4N%4B.JNL. The total length of the filename cann't exceed 256 characters.The timestamp is a system 10 digits valid time numeric data, and the <DB_BKFRM> may include both text constants and format sequences (e.g., escape sequences), that represent special character strings.

An incremental backup file name must consist of at least three special character strings: the full backup id, the database name, and the backup identification number. Backup Server assigns a full backup ID when naming incremental files in a backup sequence. When restoring a database, DBMaker uses the full backup ID to recreate the backup sequence. The database name identifies which database the incremental backup file belongs to. The backup identification number identifies the relative position of the incremental backup file in the backup sequence.

Format sequences have three parts: the escape character, the length value, and the format character. Valid format sequences are:

%[***x***]**F** — The full backup ID. The variable *x* may have values 1 through 4 where the values represent the following formats;

1: full backup id shown as YYYYMMDD, e.g., 20010917

2: full backup id shown as MMDD, e.g., 0917

3: full backup id shown as MMDDhhmm, e.g., 09171305

4: full backup id shown as DDhhmmss, e.g., 17130558

%[***n***]**B** — backup identification number

%[***n***]**N** — name of the database the journal file belongs to

The escape character identifies the start of the format sequence, and is represented by the % symbol. If you want to include the % symbol as a text constant in the backup filename format, you must use two % symbols together (i.e., %%). A single digit or one of the valid format characters shown above must immediately follow the %

symbol. If any other characters follow the % symbol the backup filename format is invalid, and DBMaker will return an error.

The length value *n* is an integer value between one and nine that determines the length of the character string generated by the format sequence. If the format sequence returns a string that can be represented in fewer characters than the length value provides then zeros will be appended to it. The database name has zeroes added to the right of the name, while all other values have zeroes added to the left. If the format sequence returns a string that requires more characters than the length value provides, it will be truncated. The database name is truncated from the right, while all other values are truncated from the left. The square brackets enclosing the length value indicate the length value is optional; do not include the square brackets when entering the format sequence. If you do not provide a value for the length, Backup Server will use the full length of the character string generated by the format sequence.

The format character identifies the type of special character string the format sequence will return. The format character must be either F, B or N; using any other character will result in an invalid backup filename format, and DBMaker will return an error. A valid format character that does not immediately follow either the escape character or the escape character and a single digit will be treated as a text constant.

The values for the date and time are taken from the system, and will only be correct if the system date and time are correct. The value for the backup identification number is the ordinal position of the backup journal file in the backup sequence. DBMaker automatically provides this number for each journal file that is backed up by Backup Server.

DBMaker provides several different methods to set the backup filename format. The method you choose depends on whether you are more comfortable editing the configuration file directly or using the JServer Manager graphical utility.

USING DMCONFIG.INI TO SET BACKUP FILE NAME FORMAT

If the database is offline, you can set the backup filename format used by Backup Server directly using the **DB_BkFrm** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will apply this backup filename format to all

backup journal files. If the database is online, changing the value of the **DB_BkDir** keyword will have no effect until the database is shut down and restarted.

➲ **To set the backup file format using the dmconfig.ini configuration file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor
2. Locate the database configuration section for a database.
3. Change the value of the **DB_BkFrm** keyword to a string containing the format to use for the backup filename format.

   NOTE *The string may contain any valid format sequences and text constants, but the total length of the resulting filename must not exceed 256 characters in length.*
4. Restart the database to begin using the new backup filename format.

USING DMSQL TO SET BACKUP FILE NAME FORMAT

The procedure **SetSystemOption** can be used to change the backup filename format while the database is running. The general syntax for the command is:

➲ **Syntax**

```
CALL SETSYSTEMOPTION('bkfrm', 'name')
```

➲ **Example**

To change the backup filename format to I1234567890_%2F%4N%4B.JNL, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bkfrm', 'I1234567890_%2F%4N%4B.JNL');
```

USING JSERVER MANAGER TO SET BACKUP FILE NAME FORMAT

You can set the backup filename format, used by Backup Server, with JServer Manager regardless of the database's offline or online status. JServer Manager automatically changes the value of the **DB_BkFrm** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server applies this backup filename format to all backup journal files.

➲ **To set the backup file format using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. Enter a format, following the for backup journal files in the **Backup File Format** field.
7. Click the **Save** button.
8. Click the Cancel button to return to the **Start Database** window

NOTE *If the **DB_BkFrm** keyword is not present in the database configuration section for the database you want to set the backup directory for, JServer Manager will add it automatically.*

Setting Multiple Backup Paths

DBMaker also supports multiple backup file paths for users. This function is useful when a user tries to save to a backup path, but the backup path does not provide enough space for the backup to be completed. If the multiple backup option is set DBMaker will then shunt the remaining data to be backed up to secondary backup locations so that the backup can be properly performed. Users are able to use multiple backup paths on full, differential or incremental backups. DBMaker has the following constraints when backing up information using multiple backup paths:

- When a database system attempts to backup files, it will try to store files in the paths one by one for each file. For example, when storing a file to backup directory 1 and the directory does not have enough space to store the file, then the file is shunted to backup directory 2, and so on. If all backup directories are full an error message will be returned.
- Only one backup directory can be used to backup files on the slave sites

- FOs must backup in the first backup directory
- The maximum number of backup paths is 32

➲ **Example**

When setting multiple backup paths DBMaker conforms to the following structure:

```
DB_BkDir = <BKDIR 1> <SIZE 1> < BKDIR 2> <SIZE 2> < BKDIR 3> <SIZE 3>…

< BKDIR n > : the n's backup path
< SIZE n > : the size of the n's backup path
```

So when setting multiple backup paths for the database **DB1** you need to set the paths in **DB_BkDir**.

```
DB_BkDir = /home/usr/dbmaker/bk 5000 /home2/backup 1000
```

When the available space in */home/usr/dbmaker/bk* is full the database will backup at */home2/backup*.

Backup Directory

The backup directory specifies where the Backup Server will place backup files. DBMaker supports single backup file path and multiple backup file paths for users. Backup Server will automatically create BkDir. However, you should choose one or more backup directory on a different disk than the database files to prevent the loss of both the database and the backup files in the event of a media error.

The backup directory is specified by the **DB_BkDir** keyword in the **dmconfig.ini** file. The value of the **DB_BkDir** keyword may contain either a full or a relative path to the backup directory. If you do not specify a backup directory, the Backup Server will automatically create a default backup directory named **backup** under the database directory. The database directory is specified by the **DB_DbDir** keyword in the **dmconfig.ini** file. The total length of the backup directory path must not exceed 256 characters in length.

However, if DBMaker database is running on replication mode (master or salve), only the sigle **BkDir** directory can be used. If you set BKDIR multi-path, the only first is used and path size is ignored. Furthermore, it is not a good idea to allow the Backup Server to create and use the default backup directory if you have more than one

database in the same directory. In this case, the backup history information from one database may overwrite or append to the backup history information from another database, rendering one or both of the backups unusable. To avoid this type of problem you can put each database in a different database directory, or explicitly specify a backup directory for each database. Placing each database in a different database directory is the preferred method, since this allows you to see exactly which files belong to which database.

DBMaker provides several different methods to set the backup directory. The method you choose depends on whether your database is online or offline, and whether you are more comfortable editing the configuration file directly or using the JServer Manager graphical utility.

USING DMCONFIG.INI TO SET BACKUP DIRECTORY

If the database is offline, you can set the backup directory used by Backup Server directly using the **DB_BkDir** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this directory as the backup directory. If the database is online, changing the value of the **DB_BkDir** keyword will have no effect until the database is shut down and restarted.

➲ **To set the backup directory using the dmconfig.ini configuration file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database.
3. Change the value of the **DB_BkDir** keyword to a string containing the name of an existing directory to set the backup directory.
4. Restart the database to begin using the new backup directory.

USING DMSQL TO SET BACKUP DIRECTORY ON LINE

The procedure SetSystemOption can be used to change the backup directory while the database is running.

➲ **Syntax**

```
CALL SETSYSTEMOPTION('bkdir', 'path')
```

Where *path* is the full path of the new backup directory. The length of the string in *path* should not exceed 256 characters.

➲ **Example**

To change the directory path to *E:/storage/database/backup/WebDB*, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bkdir', 'E:/storage/database/backup/WebDB');
```

USING JSERVER MANAGER TO SET BACKUP DIRECTORY

If the database is offline, you can set the offline backup directory used by Backup Server using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkDir** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this directory as the backup directory. If the database is online, JServer Manager can change the backup directory immediately with Run Time Setting or delay the change until the next time you restart the database when the database making an interactive backup. In either case, JServer Manager will also make a copy of the backup history file in the new backup directory.

➲ **To set the backup directory while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. Enter a path into or select the browse button ... next to the **Directory of Backup Files** field to indicate the location of the backup directory.
7. Click the **Save** button.
8. Click the Cancel button to return to the **Start Database** window.

➲ **To set the backup directory while online using JServer Manager:**

1. Select **Run Time Setting** from the database drop down menu.

2. The **Backup** page of the **Run Time Setting** window appears.
3. Select a database from the **Database Name** drop-down list box.
4. The **Login** dialog box is displayed.
5. Click **OK**, the database you logged into appears in the **Database Name** field of the **Run Time Setting** dialog box.
6. To use the updated settings in the next session, make sure that the **write to dmconfig.ini** check box is enabled.
7. To allow the updated settings to apply to the current session only, clear the checkmark in the **dmconfig.ini** check box.
8. Enter a path into or select the browse button ... next to the **Backup Directory** field to indicate a location for the backup files to be copied to.
9. Select **OK** from the bottom of the **Run Time Settings** window.

NOTE *If the **DB_BkDir** keyword is not present in the database configuration section for the database you want to set the backup directory for, JServer Manager will add it automatically.*

Setting the Old Directory

The old directory is one directory or a group of directories (up to 32), and it is used to saving a backup sequence which is one just before the last one. You should choose it on a different disk than the database files to prevent the loss of both the database and the backup files in the event of a media error.

The old directory is specified by the **DB_BkOdr** keyword in the **dmconfig.ini** file. If you do not specify it, the Backup Server will discard the previous backup sequence.

USING DMCONFIG.INI TO SET THE OLD DIRECTORY

You can set the old directory used by Backup Server directly using the **DB_BkOdr** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this directory as the old directory. If the database is online, changing the value of the **DB_BkOdr** keyword will have no effect until the database is shut down and restarted.

USING DMSQL TO SET THE OLD DIRECTORY ON LINE

The procedure **SetSystemOption** can be used to change the old backup directory while the database is running. The general syntax for the command is:

```
CALL SETSYSTEMOPTION ('bkodr', 'path')
```

Path is the full path of the new old backup directory. The length of the string in *path* should not exceed 256 characters.

➲ **Example**

To change the old directory path to *E:/storage/database/backup/WebDB*, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION ('bkodr', 'E:/storage/database/backup/WebDB');
```

USING JSERVER MANAGER TO SET THE OLD DIRECTORY

If the database is offline, you can set the location for the previous backup using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkOdr** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this directory as the backup directory. If the database is online, JServer Manager can change the old backup directory immediately or delay the change until the next time you restart the database.

➲ **To set the old backup directory while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click **Setup**. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. Enter a path into or select the browse button ... next to the **Directory of Previous Full Backup** field to indicate a location for the last full backup files to be copied to.
6. Click the **Save** button.
7. Click the Cancel button to return to the **Start Database** window.

NOTE *If the **DB_BkOdr** keyword is not present in the database configuration section for the database you want to set the backup directory for, JServer Manager will add it automatically.*

Differential Backup Settings

The differential backup schedule specifies times when Backup Server performs online differential backups. The schedule time is composed of two parts: an initial backup time and an interval time. The initial backup time specifies the date and time Backup Server will perform the first differential backup. The interval time specifies the time to wait between subsequent differential backups.

The initial full backup time is set by the **DB_FBkTm** keyword in the **dmconfig.ini** file. You must enter the value of the **DB_FBkTm** keyword as a date and time in the format YY/MM/DD HH:MM:SS. There is no default value for the initial backup time. However, when using JServer Manager to enable Backup Server, a default value is set in the **dmconfig.ini** file.

The interval time is specified by the **DB_DBkTv** keyword and is found in the **dmconfig.ini** file. The first differential backup is performed at **DB_FBkTm** + **DB_DBkTv**. You must enter the **DB_DBkTv** keyword as a time interval in the format D-HH:MM:SS. There is no default value. However, when using JServer Manager to enable Backup Server, a default value of 1-00:00:00 is set in the **dmconfig.ini** file.

The keyword **DB_DbKmx** specifies the maximum number of differential backups to retain after a full backup. Backup Server removes the oldest differential backup when the number of differential backups after a full backup exceeds **DB_DbKmx**.

USING DMCONFIG.INI TO CHANGE DIFFERENTIAL BACKUP SETTINGS

A backup schedule can set using the Backup Server when a database is offline. This can be directly set using the **DB_FBkTm** and **DB_DBkTv** keywords found in the **dmconfig.ini** configuration file. The next time you start the database, Backup Server will use these settings for the differential backup schedule. If the database is online,

changing the value of the **DB_FBkTm** and **DB_DBkTv** keywords have no effect until the database is shut down and restarted.

➲ **To set the backup schedule using the dmconfig.ini file:**

1. Open the **dmconfig.ini** file on the database server using an ASCII text editor.
2. Locate the database configuration section and then the backup schedule portion.
3. Change the value of keyword **DB_FBkTm** to a date and time value using this format YY/MM/DD HH:MM:SS.
4. Change the value of keyword **DB_DBkTv** to an interval value using this format ndays-HH:MM:SS.
5. Restart the database to activate the new backup schedule.

USING DMSQL TO CHANGE DIFFERENTIAL BACKUP SETTINGS

The procedure SetSystemOption can be used to activate the backup server. The command is:

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR','1');
```

When Backup Server is activated, calling system stored procedure **SetSystemOption** notifies it to perform a differential backup.

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3')
```

The syntax to change a differential backup interval is:

```
CALL SETSYSTEMOPTION('dbktv', 'Interval')
```

USING JSERVER MANAGER TO CHANGE DIFFERENTIAL BACKUP SETTINGS

If the database is offline, the differential backup schedule can be set using JServer Manager graphical utility. JServer Manager automatically changes the value of the **DB_FBkTm** and **DB_DBkTv** keywords found in the **dmconfig.ini** file. The next time the database is started. Backup Server uses these settings as the new differential backup schedule. If the database is online, JServer Manager can change the backup schedule immediately or delay activating the change until the next time the database is restarted. For directions on setting differential backup schedules using JServer Manager, refer to the *JServer Manager User's Guide*.

Incremental Backup Settings

The incremental backup schedule specifies the times when Backup Server will perform an online incremental backup. The schedule is composed of two parts: the initial backup time and the interval time. The initial backup time determines the date and time Backup Server will perform the first incremental backup, and the interval time determines the length of time to wait between subsequent incremental backups.

You can combine the incremental backup schedule with the journal trigger value to backup your database both on a regular schedule and when journal files fill to a specified percentage. If you do not specify an incremental backup schedule, Backup Server will not backup the database on a regular schedule.

The initial backup time is specified by the **DB_BkTim** keyword in the **dmconfig.ini** file. You must enter the value of the **DB_BkTim** keyword as a date and time in the format YY/MM/DD HH:MM:SS. There is no default value for the initial backup time.

The interval time is specified by the **DB_BkItv** keyword in the **dmconfig.ini** file. You must enter the value of the **DB_BkItv** keyword as a time interval in the format D-HH: MM:SS. There is no default value for the interval time. However, if you use JServer Manager to enable Backup Server, JServer Manager will provide a default value of 1-00:00:00 for you and write this value into the **dmconfig.ini** file.

DBMaker provides several different methods to set the incremental backup schedule. The method you choose depends on whether your database is online or offline, and whether you are more comfortable editing the configuration file directly or using the JServer Manager graphical utility.

USING DMCONFIG.INI TO CHANGE INCREMENTAL BACKUP SETTINGS

If the database is offline, you can set the backup schedule used by Backup Server directly using the **DB_BkTim** and **DB_BkItv** keywords in the **dmconfig.ini** configuration file. The next time you start the database, Backup Server will use these settings for the incremental backup schedule. If the database is online, changing the

value of the **DB_BkTim** and **DB_BkItv** keywords will have no effect until the database is shut down and restarted.

➲ **To set the backup schedule using the dmconfig.ini file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to change the backup schedule.
3. Change the value of the **DB_BkTim** keyword to a date and time using the YY/MM/DD HH:MM:SS value format.
4. Change the value of the **DB_BkItv** keyword to a time interval using the DDDDD-HH:MM:SS value format.
5. Restart the database to begin using the new backup schedule.

USING DMSQL TO CHANGE INCREMENTAL BACKUP SETTINGS

The procedure SetSystemOption can be used to change the incremental backup start time and interval while the database is running. The general syntax to change the incremental backup start time is:

```
CALL SETSYSTEMOPTION('bktim', 'StartTime')
```

The general syntax to change the incremental backup interval is:

```
CALL SETSYSTEMOPTION('bkitv', 'Interval')
```

StartTime is the time to start the first incremental backup, and has the format YY:MM:DD HH:MM:SS. *Interval* is the time interval that incremental backups occur, and has the format D-HH:MM:SS.

When Backup Server is activated, calling system stored procedure **SetSystemOption** notifies it to perform an incremental backup.

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','2')
```

➲ **Example**

To set the incremental backup interval to 1 hour, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bkitv', '0-1:00:00');
```

USING JSERVER MANAGER TO CHANGE INCREMENTAL BACKUP SETTINGS

If the database is offline, you can set the incremental backup schedule used by Backup Server using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkTim** and **DB_BkItv** keywords in the **dmconfig.ini** file. The next time you start the database, Backup Server will use these settings as the new incremental backup schedule. If the database is online, JServer Manager can change the backup schedule immediately or delay the change until the next time you restart the database. For directions on setting incremental backup schedules using JServer Manager, refer to the *JServer Manager User's Guide*.

➲ **To set the backup schedule while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. Indicate a date and a time in the **Start Time of Incremental Backup** time fields.
7. Enter the number of days, hours, minutes, and seconds between each successive full backup in the **Time Interval to Start Incremental Backup** time fields.
8. Click the **Save** button.
9. Click the Cancel button to return to the **Start Database** window.

➲ **To set the backup schedule while online using JServer Manager:**

1. Select **Run Time Setting** from the database drop down menu.
2. The **Run Time Setting** window appears.
3. Select a database from the **Database Name** drop-down list box.
4. The **Login** dialog box is displayed.
5. Click **OK**, the database you logged into appears in the **Database Name** field of the **Run Time Setting** dialog box.

6. To use the updated settings in the next session, make sure that the **write to dmconfig.ini** check box is enabled.

7. To allow the updated settings to apply to the current session only, clear the checkmark in the **dmconfig.ini** check box.

8. Indicate a date and a time for incremental backups to begin in the **Begin Time** fields.

9. Enter the number of days, hours, minutes, and seconds between each successive incremental backup in the **Interval_Time** time fields.

10. Select **OK** from the bottom of the **Run Time Settings** window.

Journal Trigger Value Settings

The journal trigger value specifies the percentage a journal file must fill before Backup Server will perform an online incremental backup. You can combine the journal trigger value with the backup schedule to backup your database on a regular schedule and when journal files fill to the specified percentage.

The journal trigger value is specified by the **DB_BkFul** keyword in the **dmconfig.ini** file. The value of the **DB_BkFul** keyword may be an integer value in the range 50 through 100, or zero. Values between 50 and 100 represent the percentage a journal file must fill before Backup Server performs a backup. A value of zero causes Backup Server to perform a backup whenever a journal file fills completely. Setting the value to 0 is effectively the same as setting it to a value of 100, since both will cause Backup Server to perform a backup whenever a journal file fills completely (e.g., 100% is full). If you do not specify a value for the journal trigger value, Backup Server will use the default value of ***90***.

DBMaker provides several different methods to set the journal trigger value. The method you choose depends on whether your database is online or offline, and whether you are more comfortable editing the configuration file directly or using the JServer Manager graphical utility.

USING DMCONFIG.INI TO CHANGE THE JOURNAL TRIGGER VALUE

If the database is offline, you can set the journal trigger value used by Backup Server directly using the **DB_BkFul** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this setting for the journal trigger value. If the database is online, changing the value of the **DB_BkFul** keyword will have no effect until the database is shut down and restarted.

➲ **To set the journal trigger value using the dmconfig.ini file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to change the journal trigger value.
3. Change the value of the **DB_BkFul** keyword to an integer value between 50 and 100, or set it to zero.
4. Restart the database to begin using the new journal trigger value.

USING DMSQL TO CHANGE JOURNAL TRIGGER VALUE

The procedure SetSystemOption can be used to change the journal trigger value while the database is running. The general syntax to change the incremental backup start time is:

```
CALL SETSYSTEMOPTION('bkful', 'n')
```

Where *n* is either 0 or 50 through 100. Setting *n* to 0 will trigger the backup server whenever a journal file is full. Setting *n* to a value between 50 and 100 specifies the percentage a journal file fills to before the backup server activates.

➲ **Example**

To set the journal trigger value to 75 percent, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bkful', '75');
```

USING JSERVER MANAGER TO CHANGE THE JOURNAL TRIGGER VALUE

If the database is offline, you can set the journal trigger value used by Backup Server using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkFul** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this setting as the new journal trigger value. If the database is online, JServer Manager can change the journal trigger value immediately or delay the change until the next time you restart the database. For directions on how to set the journal trigger value using JServer Manager, refer to the *JServer Manager User's Guide*.

➲ **To set the journal trigger value while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. Incremental backups can be set to automatically execute when journal files have filled to a set percentage. Select one of the following:
   - Select the **Backup when any Journal File is Full** option button to set incremental backups to execute when any journal file is filled.
   - Enter a value from 50 to 100 in the **% Full** field to set incremental backups to execute when any journal file is filled to the value entered.
6. Click the **Save** button.
7. Click the Cancel button to return to the **Start Database** window.

➲ **To set the journal trigger value while online using JServer Manager:**

1. Select **Run Time Setting** from the database drop down menu.
2. The **Run Time Setting** window appears.
3. Select a database from the **Database Name** drop-down list box.
4. The **Login** dialog box is displayed.

5. Click **OK**, the database you logged into appears in the **Database Name** field of the **Run Time Setting** dialog box.

6. To use the updated settings in the next session, make sure that the **write to dmconfig.ini** check box is enabled.

7. To allow the updated settings to apply to the current session only, clear the checkmark in the **dmconfig.ini** check box.

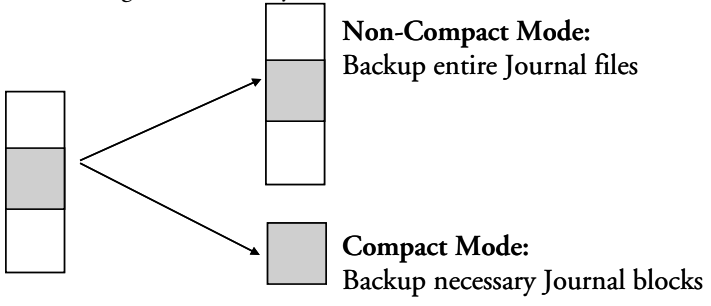
8. Incremental backups can be set to automatically execute when journal files have filled to a set percentage. Next to **Journal Full Percentage**:

   - Select the **Use Default Value** option button to set incremental backups to execute when any journal file is completely filled.
   - Enter a value from 50 to 100 in the **50 – 100 %** field to set incremental backups to execute when any journal file is filled to the value entered.

9. Select **OK** from the bottom of the **Run Time Settings** window.

Compact Backup Mode Settings

Compact backup mode specifies whether Backup Server will backup entire journal files or only full journal blocks when it performs an online incremental or differential backup. This is possible since not every journal block contains data needed to restore a database, so Backup Server will only backup the necessary journal blocks when it performs a backup. This allows you to save storage space on your backup device, but it also means restoring a database may take more time.



The compact backup mode setting is specified by the **DB_BkCmp** keyword in the **dmconfig.ini** configuration file. The value of the **DB_BkCmp** keyword may be zero

or one. Setting the value to one enables compact backup mode, and setting it to zero disables compact backup mode. If you do not specify a value for the compact backup mode, Backup Server will use the default value of ***one*** (enabled).

DBMaker provides several different methods to set the compact backup mode. The method you choose depends on whether your database is online or offline, and whether you are more comfortable editing the configuration file directly or using the JServer Manager graphical utility.

USING DMCONFIG.INI TO SET COMPACT BACKUP MODE

If the database is offline, you can set the compact backup mode setting used by Backup Server directly using the **DB_BkCmp** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this setting for the compact backup mode. If the database is online, changing the value of the **DB_BkCmp** keyword will have no effect until the database is shut down and restarted.

➲ **To set the Compact Backup Mode using the dmconfig.ini configuration file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to change the compact backup mode.
3. Change the value of the **DB_BkCmp** keyword to one to enable compact backup mode, or zero to disable compact backup mode.
4. Restart the database to begin using the new compact backup mode.

USING DMSQL TO SET COMPACT BACKUP MODE

The **SetSystemOption** procedure can be used to change the compact backup mode while the database is running. A successful backup does not require every journal block in a journal file. If this keyword **DB_BKCMP** is set to 1 the backup server will only back up the journal blocks that require backup. The command is:

```
dmSQL> CALL SETSYSTEMOPTION ('bkcmp', '1');
```

USING JSERVER MANAGER TO SET COMPACT BACKUP MODE

If the database is offline, you can set the compact backup mode setting used by Backup Server using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkCmp** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this setting as the new compact backup mode setting. If the database is online, JServer Manager can change compact backup mode setting immediately or delay the change until the next time you restart the database. For directions on how to set the Compact Backup Mode using JServer Manager, refer to the *JServer Manager User's Guide*.

➲ **To set the Compact Backup Mode while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. To enable compact backup, Click the **Enable Compact Backup** check box
7. Click the **Save** button.
8. Click the Cancel button to return to the **Start Database** window.

➲ **To set the Compact Backup Mode while online using JServer Manager:**

1. Select **Run Time Setting** from the database drop down menu.
2. The **Run Time Setting** window appears.
3. Select a database from the **Database Name** drop-down list box.
4. The **Login** dialog box is displayed.
5. Click **OK**, the database you logged into appears in the **Database Name** field of the **Run Time Setting** dialog box.
6. To use the updated settings in the next session, make sure that the **write to dmconfig.ini** check box is enabled.

7. To allow the updated settings to apply to the current session only, clear the checkmark in the **dmconfig.ini** check box.

8. To enable compact backup, click the **Use Compact Backup Mode** check box.

9. Select **OK** from the bottom of the **Run Time Settings** window.

Full Backup Schedule

The full backup schedule specifies the times when Backup Server will perform an online full backup. The schedule is composed of two parts: the initial backup time and the interval time. The initial backup time determines the date and time Backup Server will perform the first full backup, and the interval time determines the length of time to wait between subsequent full backups.

You can combine full or differential backup schedules with an incremental backup schedule to backup your database. If you do not specify a full backup schedule, Backup Server does not perform full backups on a regular schedule.

The initial backup time is specified by the **DB_FBkTm** keyword in the **dmconfig.ini** file. You must enter the value of the **DB_FBkTm** keyword as a date and time in the format YY/MM/DD HH:MM:SS. There is no default value for the initial backup time.

The interval time is specified by the **DB_FBkTv** keyword in the **dmconfig.ini** file. Enter the value of the **DB_FBkTv** keyword as a time interval in the format D-HH:MM:SS. There is no default value for the interval time.

Lastly, the keyword **DB_BkChk** specifies whether check database before full backup and differential backup and the keyword **DB_BkRTs** specifies whether the backup server includes the read-only tablespace files when performing a full-backup. To enable or disable the two features, you can set the keyword **DB_BkChk** and **DB_BkRTs** in **dmconfig.ini**, or change BKCHK and BKRTS with the system procedure **SetSystemOption** while the database is running.

USING DMCONFIG.INI TO SET THE FULL BACKUP MODE

If the database is offline, you can set the full backup schedule used by Backup Server directly using the **DB_FBkTm** and **DB_FBkTv** keywords in the **dmconfig.ini** file. The next time you start the database, Backup Server will use these settings for the full backup schedule. If the database is online, changing the value of the **DB_FBkTm** and **DB_FBkTv** keywords will have no effect until the database is shut down and restarted.

➲ **To set the full backup schedule using the dmconfig.ini configuration file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to change the full backup schedule.
3. Set the configuration parameter **DB_FBkTm** to a value of the format YY/MM/DD HH:MM:SS, and **DB_FBkTv** to a value of the format D-HH:MM:SS.
4. Restart the database to begin using the new full backup schedule.

USING DMSQL TO SET THE FULL BACKUP SCHEDULE

The procedure **SetSystemOption** can be used to change the full backup start time and interval while the database is running. The general syntax to change the full backup start time is:

```
CALL SETSYSTEMOPTION('fbktm', 'StartTime')
```

The general syntax to change the full backup interval is:

```
CALL SETSYSTEMOPTION('fbktv', 'Interval')
```

StartTime is the time to start the first full backup, and has the format YY:MM:DD HH:MM:SS. *Interval* is the time interval that full backups occur, and has the format D-HH:MM:SS.

When Backup Server is activated, call the system stored procedure **SetSystemOption** to initiate an full backup.

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1');
```

➲ **Example**

To set the full backup interval to 1 hour, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('fbktv', '0-1:00:00');
```

USING JSERVER MANAGER TO SET THE FULL BACKUP MODE

You can set the full backup schedule with the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_FBkTm** and **DB_FBkTv** keywords in the **dmconfig.ini** file. The next time you start the database, Backup Server will use this setting as the new full backup schedule.

➲ **To set the full backup schedule using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To start the backup server, select the **Start Backup Server** check box.
6. Indicate a date and a time in the **Start Time of Full Backup** time fields.
7. Enter the number of days, hours, minutes, and seconds between each successive full backup in the **Full Backup Daemon Interval** days and time fields.
8. Click the **Save** button.
9. Click the **Cancel** button to return to the **Start Database** window.

Backup Mode of File Objecta

The file object backup mode lets the database administrator decide whether Backup Server will back up file objects during a full backup. It is also possible to specify Backup Server to back up just system file objects or system and user file objects.

It is possible to set the file object backup mode in a number of ways. The configuration keyword **DB_BkFoM** determines the setting during database startup,

but it may also be modified during runtime with dmSQL or the JServer Manager utility.

The backup server will move all files from the previous backup to the old backup directory specified by **DB_BkOdr**.

Starting file object backup will cause the database to require more time to complete a full backup, depending on how many file objects are in the database. The total cost of a complete full backup includes (1) copying the previous full backup if **DB_BkOdr** is set; (2) copying all database files; (3) copying all journal files; and (4) copying all file objects if **DB_BkFoM** is set. Be sure that enough disk space is available in the backup directory specified by **DB_BkDir** (and **DB_BkOdr** if applicable) for all mentioned backup files to avoid backup failure.

File objects are copied into an FO directory that is created in the backup directory at the time a full backup is performed. File objects are renamed sequentially when they are copied to the backup file object directory. The files in the **/FO** subdirectory are renamed starting with the letters FO followed by a ten digit serial number. All backup file objects are appended with the file extension .BAK. The mapping between the source file name and path, and the backup file name is recorded in the object-mapping file, i.e., **dmFoMap.his**.

BACKUP FILE OBJECT MAPPING FILE

The file object mapping file dmFoMap.his is created in the "*DB_BkDir/ FO*" directory. It is a pure ASCII text file that records the original external file name and backup file name.

➲ **Syntax**

```
Database Name: MYDB
Begin Backup FO Time: 2001.5.13 2:33
FO Backup Directory: /DBMaker/mydb/backup/FO (i.e., DB_BkDir/FO)
[Mapping List]
s, fo0000000000.bak, "/DBMaker/mydb/fo/ZZ000001.bmp"
u, fo0000000001.bak, "/home2/data/image.jpg"
....
s, fo0000002345.bak, "/DBMaker/mydb/fo/ZZ00AB32.txt"
```

The content before "[Mapping List]" is only a description for user reference. Each line after "[Mapping List]" represents a record that shows the file object type (s = system

file object, u = user file object), the new file in */FO* subdirectory and its original file name and path. This mapping file is necessary for restoration of file objects.

USING DMCONFIG.INI TO SET BACKUP MODE OF FILE OBJECTS

The configuration file keyword **DB_BkFoM** determines the file object backup mode:

DB_BkFoM = 0: Not backup file objects

DB_BkFoM = 1: Backup system file objects only

DB_BkFoM = 2: Backup both system and user file objects

For backup file object mode (**DB_BkFoM** = 1, 2), the backup server will copy all file objects to the "*/fo*" subdirectory under the backup directory. The schedule follows the full backup schedule.

➲ **Example**

An entry in a **dmconfig.ini** file for specifying the file object backup parameters looks like this.

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 01/05/01 00:00:00        ; begins at midnight, May 1, 2001.
DB_FBkTv = 1-00:00:00               ; interval is once every day.
DB_BkDir = /home/dbmaker/backup     ; backup directory
DB_BkFoM = 2                        ; backup both system and user file objects
```

Since the backup mode is 2, the backup server will copy all external files (user file objects) and system file objects to the */home/dbmaker/backup/FO* directory. If the **FO** subdirectory does not exist, the Backup Server will create it.

USING DMSQL TO SET BACKUP MODE OF FILE OBJECTS

The procedure SetSystemOption can be used to change the file object backup mode while the database is running. The general syntax to change the file object backup mode is:

```
CALL SETSYSTEMOPTION('bkfom', 'n')
```

Where *n* is 0, 1, or 2. Setting *n* to 0 will turn the file object backup mode to off. Setting *n* to 1 configures backup server to back up all system file objects during a full

backup. Setting *n* to 2 configures backup server to back up all system and user file objects during a full backup.

➲ **Example**

To configure Backup Server to perform a full backup on all user and system file objects, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bkfom', '2');
```

USING JSERVER MANAGER TO SET BACKUP MODE OF FILE OBJECTS

The settings under the **Backup File Object Mode** effect how file objects are copied during the full backup process. Selecting **Do Not Backup File Objects** disables file backup during the full backup process. Selecting **Backup System File Objects Only** will result in system file objects being backed up during automatic full backups. Selecting **Backup System and User File Objects** will result in both system file objects and user file objects being copied to the backup directory during automatic full backups.

➲ **To set the file object backup mode during database startup:**

1. Click **Setup** in the **Start Database** window. The **Start Database Advanced Settings** window appears.
2. Click the **Backup** tab in the **Start Database Advanced Settings** window.
3. Enable the **Start Backup Server** check box.
4. To enable full backups to be performed by the backup server.
   a) Enter a path or select the browse button ... next to the **Directory of Backup Files** field to indicate the location of the backup directory.
   b) Indicate a date and a time in the **Start Time of Full Backup** time fields.
   c) Enter the number of days, hours, minutes, and seconds between each successive full backup in the **Full Backup Daemon Interval** time fields.
5. To select what types of file objects are backed up during the backup process:
   a) Select Do not backup file objects to prevent file objects from being backed up.
   b) Select Backup system file objects only to only back up system file objects.

c) Select Backup system and user file objects to back up all file objects.

6. Click **Save**.

7. Click **Cancel** to return to the **Start Database** window, and click **Start** to start the database.

Backup Mode of Stored Procedures

The Backup Mode of stored procedures lets the database administrator decide whether Backup Server will back up ESQL stored procedures and JAVA stored procedures during a full backup.

It is possible to set the Backup Mode of stored procedures in a number of ways. The configuration keyword **DB_BkSPm** determines the setting during database startup, but it may also be modified during runtime with dmSQL or the JServer Manager utility.

The backup server will move the previous backup of stored procedures to the old backup directory specified by **DB_BkOdr**.

Starting stored procedures backup will cause the database to require more time to complete a full backup, depending on how many file objects are in the database. The total cost of a complete full backup includes (1) copying the previous full backup if **DB_BkOdr** is set; (2) copying all database files; (3) copying all journal files; (4) copying all file objects if **DB_BkFoM** is set; (5) copying all ESQL stored procedures and JAVA stored procedures if **DB_BkSPm** is set. Also, ensure that there is enough disk space in the backup directory specified by **DB_BkDir** for all backup files to avoid backup failure.

Stored procedures are copied into the subdirectory **SP** that is created in the backup directory at the time a full backup is performed. Stored procedures are renamed when they are copied to the directory for backed-up stored procedures. The backup information about copied stored prcedures is recorded in the file **dmSpBk.his**.

BACKUP STORED PROCEDURE INFORMATION FILE

The backup information is composed of three parts, database name, backup time and backup list used to record which rows have been backed up. The backup information file **dmSpBk.his** is created in the subdirectory **SP** under the directory specified by **DB_BkDir**. It is a pure ASCII text file that records backup information about the copied stored procedure. The format looks like:

```
Database Name: MYDB
Begin Backup SP time: 2014/06/20 09:13:06
[Backup List]
ESQLSP, SYSADM, A4, A4SYSADM.dll, A4SYSADM.ec
....
JAR, "", "", employee.jar, ""
```

The content before "[Backup List]" is only a description for user reference. Each line after "[Backup List]" represents a record that shows stored procedures' type, stored procedure's owner, stored procedures' name, the name of stored procedures' object files and the name of its corresponding source files. This backup information file is necessary for restoration of stored procedures.

USING DMCONFIG.INI TO SET BACKUP MODE OF STORED PROCEDURES

The file keyword **DB_BkSPm** determines the backup mode of stored procedures:

- **DB_BkSPm** = 0: Do not back up ESQL stored procedures and JAVA stored procedures
- **DB_BkSPm** = 1: Back up all ESQL stored procedures and JAVA stored procedures

➲ **Example**

An entry in a **dmconfig.ini** file for specifying the stored procedure backup parameters.

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 14/05/01 00:00:00       ; begins at midnight, May 1, 2014
DB_FBkTv = 1-00:00:00               ; interval is once every day
DB_BkDir = /home/dbmaker/backup     ; backup directory
DB_BkSPm = 1              ; backup all ESQL stored procedures and JAVA stored
procedures
```

Since the value of **DB_BkSPm** is 1, the Backup Server will backup all ESQL stored procedures and JAVA stored procedures to the subdirectory **SP** under the directory specified by **DB_BkDir**. If the subdirectory **SP** does not exist, the Backup Server will create it.

USING DMSQL TO SET BACKUP MODE OF STORED PROCEDURES

The procedure **SetSystemOption** can be used to change the backup mode of stored procedures while the database is running. The general syntax to change the stored procedures is:

```
CALL SETSYSTEMOPTION('BKSPM', 'n')
```

Users can assign 0 or 1 to *n*. If assign 0 to *n*, the Backup Server will not back up ESQL stored procedure and JAVA stored procedures; if assign 1 to *n*, the Backup Server will back up all ESQL stored procedure and JAVA stored procedures during a full backup.

➲ **Example**

To configure Backup Server to perform a full backup on all stored procedures, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('BKSPM', '1');
```

USING JSERVER MANAGER TO SET BACKUP MODE OF STORED PROCEDURES

Regardless of the online status of a database, you can enable backing up ESQL stored procedures and JAVA stored procedures dynamic using the JServer Manager graphical utility. JServer Manager automatically changes the value of **DB_BkSPm** keyword in the **dmconfig.ini** configuration file. If the database is offline, the next time you start the database, this new setting will take effect. For directions on how to set the Backup Mode of stored procedures during database startup or with the **Run Time Settings** dialog in JServer Manager, please refer to the *JServer Manager User's Guide*.

Inactivate Backup Server

DBMaker will automatically start Backup Server while starting the database. Backup Server is disabled by default. You can control the state of backup server with **DB_BkSvr**. When **DB_BkSvr** is set to 0, the backup server is inactive; when **DB_BkSvr** is set to 1, the backup server is active. When you no longer want the backup server is active, you can set the value of the **DB_BkSvr** keyword to 0 in the **dmconfig.ini** file or change BkSvr with *call setsystemoption('bksvr','0')* after the database is started.

USING DMCONFIG.INI TO INACTIVATE BACKUP SERVER

If the database is offline, you can disable Backup Server directly using the **DB_BkSvr** keyword in the **dmconfig.ini** file. The next time you start the database, Backup Server will not start. If the database is online, changing the value of the **DB_BkSvr** keyword will have no effect until the database is shut down and restarted.

➲ **inactivate Backup Server using the dmconfig.ini file:**

1. Open the **dmconfig.ini** file on the database server using any ASCII text editor.
2. Locate the database configuration section for a database to change the backup mode.
3. Change the *value* of the **DB_BkSvr** keyword to 0 to disable the Backup Server.
4. Restart the database.

USING DMSQL TO INACTIVATE BACKUP SERVER

The procedure **SetSystemOption** can be used to change the state of backup server while the database is running. The general syntax to change the state of Backup Server:

```
CALL SETSYSTEMOPTION('bksvr', 'n')
```

Where *n* is 0, or 1. Setting *n* to 0 will inactivate the backup server. Setting *n* to 2 will activate the backup server.

➲ **Example**

To inactivate the backup server, enter the following line at the dmSQL command prompt.

```
dmSQL> CALL SETSYSTEMOPTION('bksvr', '0');
```

USING JSERVER MANAGER TO INACTIVATE BACKUP SERVER

If the database is offline, you can disable Backup Server using the JServer Manager graphical utility. JServer Manager will automatically change the value of the **DB_BkSvr** keyword in the **dmconfig.ini** configuration file. The next time you start the database, Backup Server will not start. If the database is online, disabling Backup Server will have no effect until the database is shut down and restarted.

➲ **To inactivate Backup Server while offline using JServer Manager:**

1. Select **Start Database** from the main console or the **Database** pull-down menu. The **Start Database** dialog box opens.
2. Select the database to modify from the **Database Name** drop-down list box.
3. Click the **Setup** button. The **Start Database Advanced Settings** window opens.
4. Click the **Backup** tab.
5. To stop the backup server, clear the **Start Backup Server** check box.
6. Click the **Save** button.
7. Click the Cancel button to return to the **Start Database** window.

9.7 Backup History Files

Automatic backups using the backup server can store the information about which journal files were backed up, when they were backed up, and where the backup files are located in the backup history file by automatically.

Locating the Backup History File

Backup history file is a text file located in the first directory of **DB_BkDir** keyword in the **dmconfig.ini** file. This file is created in the online backup path and named **dmBackup.his**. The file is automatically used during restoration of a database, but the offline backup is recorded with **offBackup.his**.

Understanding the Backup History File

Backup history files contain all information pertaining to the id number, file names, and time and date that backups were made. DBMaker uses the backup history file to track backup sequences and ensure the consistency of full, differential and incremental backups within each sequence.

The following is the backup history file format:

```
<backup_id>: file_name -> archive_file_name, time, event
```

This denotes that a file named *file_name* was copied to an archive file named *archive_file_name* at time because of event. The event is a text string indicating the reason for the backup. This string can be JOURNAL-FULL, TIME-OUT, ON-LINE-FULL-BACKUP-BEGIN, ON-LINE-FULL-BACKUP, or ON-LINE-FULL-BACKUP-END. The string JOURNAL-FULL indicates an incremental backup was performed because the journal was full. The string TIME-OUT indicates a differential or an incremental backup was performed because the scheduled backup interval elapsed. The string ON-LINE-FULL-BACKUPxxxx means it is a full backup.

Using the Backup History File

If journal full occurs frequently, lower the backup journal full percentage or shorten the time interval. Also, find out if the backup interval is too short by checking the backup history file. If the same journal file is backed up consecutively in the backup history file, the time interval may be too short. This situation will waste disk space because each file may only contain a few changed blocks. To avoid this, enable compact backup mode or lengthen the backup time interval.

If many journal files are backed up every time, it may mean the time interval is too long. This situation is more dangerous because of the possibility of losing more data when a disk fails. To avoid this, users should shorten the backup time interval.

To shorten the time of recovery from media failures, perform full backups regularly, even if you are using the backup server. In addition, this will also reduce the amount of backup storage needed.

Understanding the File Object Backup History File

The file-object backup history file, **dmFoMap.his**, keeps a record of all file objects that have been backed up by setting the file object backup configuration parameter on. **dmFoMap.his** is placed in the "<*DB_BkDir*>*FO*" directory, is a pure ASCII text file that records the original external file name and backup file name.

The following is the file format:

```
Database Name: MYDB
Begin Backup FO Time: 2001.5.13 2:33
FO Backup Directory: /DBMaker/mydb/backup/FO (i.e. DB_BkDir/FO)
[Mapping List]
s, fo0000000000.bak, "/DBMaker/mydb/fo/ZZ000001.bmp"
u, fo0000000001.bak, "/home2/data/image.jpg"
....
s, fo0000002345.bak, "/DBMaker/mydb/fo/ZZ00AB32.txt"
```

In the first column, **s** or **u** represent system or user file objects, respectively. The second column gives the backup name, and the third column gives the full name and path of the original file object.

Understanding the Stored Procedure Backup History File

The stored procedures backup history file, **dmSpBk.his**, keeps a record of all ESQL stored procedures and JAVA stored procedures that have been backed up by setting the stored procedures backup configuration parameter on. **dmSpBk.his**, placed under the subdirectory **SP** under the directory specified by **DB_BkDir**, is a pure ASCII text file that records backup information of ESQL stored procedures and JAVA stored procedures.

The following is the file format:

```
Database Name: MYDB
Begin Backup SP time: 2014/06/20 09:13:06
[Backup List]
ESQLSP, SYSADM, A4, A4SYSADM.dll, A4SYSADM.ec
....
JAR, "", "", employee.jar, ""
```

The first column gives stored procedures' type; the second column gives stored procedure's owner; the third column gives stored procedures' name; and the fourth column and the fifth column give the name of stored procedures' object files and the name of its corresponding source files respectively.

9.8 Backup on Replication Databases

On both normal database and master, but not slave database, users can do full backup, differential backup and incremental backup. The method is same as before. However, JServerManager can't do incremental backup interactively on master database. Furthermore, it can't clear incremental backup files when doing full backup interactively on master database.

Please note that on a master database, potentially,many incremental backup files ahead of a full backup is still remained in backup sequence for replication. Meanwhile the replication server may not clear incremental backup files because of full backup, so maybe a large number of files exist in **DB_BkDir** if next full backup will not be done on a long duration.

In one word, the replication sever must cooperate with the backup sever well, and they can't disturb each other. On the one hand ,backup should not damage the replication, in other words, replication server always can replication all transactions to slave sites regardless of whether a full backup or a differential backup has been done or is being done, on the other hand, replication can't damage backup sequence.

It is doable to restore the master database with backup sequence. However, after the master database restored, the database replication will not confinue. If users want to continue replicating the database, all slave databases must have been replaced by new master database, that is to say, users must copy the master database files to replace all slave databases files.

There are some constraints for backup on replication database:

- When the master database started up, BMode and BkSvr must be on.
- BMode, BkSvr, BkDir can't be changed during runtime both on master and slave databases, for example, call setsystemoption ('bkdir','new-bkdir') will return an error.

- Both on master and slave databases site, **DB_BkDir** keyword in the dmconfig.ini file should be single path. If Users set **DB_BkDir** keyword is multi-path, only the first path is used and the path size is ignored.
- On a master database, it is disable to do incremental backup interactively by JServerManager.
- On a slave database, it is disable to do full backup, differential backup and incremental backup.

9.9 Recovery Options

Restoring a database recreates the database as it existed at the time of the most recent full backup plus changes as applied by the backed up journal files.

Analyzing Options

If the database was running in NONBACKUP mode, the only option for restoration after a disk failure is to restore the most recent full backup and restart the database. All work performed since the last full backup will be lost, and must be re-entered.

If the database was running in BACKUP, BACKUP-DATA, or BACKUP-DATA-AND-BLOB mode, several recovery options are available for reconstruction.

Preparing for Restoration

To restore a database after a disk error, consider:

- Points in time to restore the database

To restore to the time when the disk error occurred, backup all journal files of the damaged database; these files will help DBMaker to restore the database to the most current time.

- Previously backed up files
- Find out where the most current full backup and all subsequent differential and incremental backups are located. For example, suppose you perform a full backup on the 30th day of every month, a differential backup every 15th day and an incremental backup every 10 days. If your system is damaged on May 25th, you need the full backup from April 30th, the differential backup from May 15th and the incremental backups from May 10th and May 20th, and the damaged journal files from May 25th. After locating these files, DBMaker can restore your database to the state it was in before the failure on May 25th. The valid backup sequence that is composed of a group of full backup files and a series of differential backup

files and incremental backup files is essential for the restoration. The online backup sequence is identified by a backup history file named **dmbackup.his** and the offline backup sequence is identified by a backup file named **offbackup.his.** This makes the backup history file especially important because DBMaker reads it to get this information when restoring a database.

Performing a Restoration

When executing the restoration process, DBMaker will do the following actions:

- Copying all full backup files; includes data files, blob files and journal files, to the directory specified by the **DB_DbDir** keyword in the **dmconfig.ini**. This operation will overwrite the original database files. So it is strongly recommended that user can manually copy original database files to other place before running the restoration tools, at least make sure the journal files to be saved, to insure that if the restoration failed, there is also another chance for the database to be restored to the most current time.
- Applying the differential or incremental backup files or both into database.

When using restoration tools, users can specify:

- Whether restore database section in system dmconfig.ini. To restore it, specify the full path of restored dmconfig.ini.
- If you want to use a backup sequence to restore the database to a location different from the original, modify the keywords in the data file *path*, these include **DB_DbDir**, **DB_DbFil**, **DB_UsrDb**, and so on. If the backup sequence is moved to another location or computer, consider the following when restoring the database:

  a) Database names in **dmconfig.ini** must be consistent with the backup database name.

  b) Set keywords BkDir and others for data and blob files as needed.

  c) The value of **DB_JnFil** must be set if there is more than one jnl file. Ensure that this value matches the number of jnl files in backup database.

d) If backup files are located in multiple folders the **DB_BkDir** keyword must be set to include all folders where backup files are located. The dmbackup.his file must be located in the first BkDir.

NOTE *Users can copy an existing dmconfig.iniconfiguration file from the folder where the backup sequence existed, then configure a new dmconfig.ini file by modifying the relevant keywords.*

- Backup full path of a backup history file dmBackup.his or offBackup.his if the dmBackup.his or offBackup.his is not located in the default directory.
- Restore time (RTime). RTime denotes what time the database to be restored to, it will determine whether the current backup sequence is available or not, and which differential and incremental backup files will be applied to database. User can specify it in restoration tools, or add keyword **DB_RTime** into system dmconfig.ini or backup dmconfig.ini which will be restored. If RTime is not specified, the default value is the current time.

DBMaker provides two methods to perform restoration. One is by JServer Manager Tool and the other is by Rollover command line tool.

For more information on usage of the JServer Manager, please refer to *JServer Manager User Guide*. And for more information on usage of the rollover command line tool, please refer to next section *Restoring database by Rollover*.

Restoring database by Rollover

User can also use the Rollover which is a command line tool to restore the database. Its principle is same as the Restore Database of JServer Manager.

The usage of rollover is like:

rollover database_name [**-i inifile**] [**-r rtime**] [**-h hisfile**] [**-m foMapfile**] [**-f FOtype**] [**-t SpResTyp**]

There are six optional parameters in the square bracket:

-i specify full path of **dmconfig.ini** If user specifies the **dmconfig.ini** to restore, rollover will replace the database section in system dmconfig.ini with the

corresponding database section in specified **dmconfig.ini**, otherwise, DBMaker will not restore **dmconfig.ini**.

-r denote the time that database should be restored to. The option –r is the first method to specify rtime, the second method is to add **DB_RTime** keyword into system dmconfig.ini or backup dmconfig.ini which will be specified to restore database. If neither –r option nor **DB_RTime** keyword, the rtime will be the current time.

-h give full path of **dmBackup.his** or **offBackup.his**. The default is *DB_BkDir|dmBackup.his* or *DB_BkDir|offBackup.his*

-m give full path of **dmFoMap.his**. The default is ***DB_BkDir|FO|dmFoMap.his***

-f specify which type FO files will be restored. There are four values, the value of 0 means no FO files to be restored; the value of 1 will restore system FO; value of 2 will restore user FO and value of 3 will restore all FO. The default value is ***3***.

-t specify whether restore stored procedures or not. There are two values, the value of 0 means no stored procedures will be restored; value of 1 will restore all stored procedures. The default value is ***1***.