



# **The Magic<sup>®</sup> Guide to SQL**

*Magic Enterprise Edition - Version 8*

---

**Magic Software Enterprises**

The information in this manual is subject to change without prior notice and does not represent a commitment on the part of MSE.

MSE makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms and conditions of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this manual and/or databases may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or information recording and retrieval systems, for any purpose other than the purchaser's personal use, without the prior express written permission of MSE.

All references made to third party trademarks are for informational purposes only regarding compatibility with the products of Magic Software Enterprises Ltd.

Unless otherwise noted, all names of companies, products, street addresses, and persons contained herein are part of a completely fictitious scenario or scenarios and are designed solely to document the use of Magic.

Magic®, Magic PC™, Magic II™, and MagicGate™ are trademarks of Magic Software Enterprises Ltd.

Btrieve® is a registered trademark of Pervasive Software, Inc.

Novell NetWare LAN® are registered trademarks of Novell.

Informix™ and C-ISAM™ are trademarks of INFORMIX.

Clipper® is a registered trademark of Computer Associates International, Inc.

dBASE®, dBASE III® and dBASE IV® are registered trademarks of Borland International, Inc.

FTP® and PC/TCP® Network Software are registered trademarks of FTP Software Inc.

IBM®, Topview™, and RISC System/6000™ are trademarks of International Business Machines Corporation.

Microsoft®, FoxPro®, Windows™, WindowsNT™, and ActiveX™ are trademarks of Microsoft Corp.

Oracle® is a registered trademark of the Oracle Corporation.

SCO® is a registered trademark of The Santa Cruz Operation, Inc.

Sybase® is a registered trademark of Sybase, Inc.

UNIX® is a registered trademark of UNIX System Laboratories.

VAX, VMS, VAX/VMS, OpenVMS, VT, Rdb, RMS, ULTRIX Connection, and DECnet are trademarks of Digital Equipment Corporation.

All company and product names are the trademarks or registered trademarks of their respective companies.

Revised January 2000

02 01 00 10 9 8 7 6 5

4183-0400-0001

Copyright • 2000, 1999, 1998 by Magic Software Enterprises Ltd. All rights reserved.

# Contents

---

## **1 Introduction and Overview**

Audience . . . . .	18
Scope . . . . .	18
Objectives of the Guide. . . . .	19
Examples . . . . .	19
SQL Structure and Features . . . . .	20
The SQL Engine . . . . .	20
DBMS Functions Controlled by SQL . . . . .	21
Basic Relational Database and SQL Concepts . . . . .	24
Elements of Database Application Development . . . . .	25
SQL Data Definition Language . . . . .	26
SQL Data Manipulation Language . . . . .	26
The SQL Database Administrator . . . . .	27
Magic University International - MUI . . . . .	28

## **2 Data Types**

ANSI/ISO Data Types . . . . .	29
Extended Data Types . . . . .	31
Data Type Limitations . . . . .	32
Data Type Differences . . . . .	32
Constants . . . . .	33

Numeric Constants . . . . .	33
Common RDBMS Data Types . . . . .	34
Summary . . . . .	35
<b>3 Connecting to a Database</b>	
SQL Servers and Databases . . . . .	37
SQL User Interface . . . . .	38
Connect String . . . . .	39
SQL Reconnect . . . . .	39
<b>4 Creating Database Objects</b>	
Creating a Database . . . . .	41
Altering Database Parameters . . . . .	42
The Data Definition Language - DDL . . . . .	42
Views . . . . .	45
<b>5 Selecting and Sorting Columns</b>	
Selecting All the Columns From a Table . . . . .	47
Selecting Some Columns From a Table . . . . .	49
How to Eliminate Duplicate Rows . . . . .	51
Sorting Query Results - the ORDER BY Clause . . . . .	53
<b>6 Row Selection - the WHERE Clause</b>	
Selecting Rows with the WHERE Clause . . . . .	57
Search Conditions . . . . .	58
<b>7 Advanced SELECT Statements</b>	
Expressions . . . . .	71
Arithmetic Expressions . . . . .	72
The NULL Functions . . . . .	73
The Concatenation Function . . . . .	74
Multi-table Queries . . . . .	75

The Group Functions . . . . .	77
The GROUP BY Clause - Grouped Queries . . . . .	78
The HAVING Clause - Group Search Conditions . . . . .	80
Nested SELECT Statements . . . . .	82
Data Conversion . . . . .	84
Programming with SQL . . . . .	87
<b>8 Modifying Data - Insert, Delete, Update</b>	
INSERT . . . . .	89
DELETE . . . . .	92
UPDATE . . . . .	93
<b>9 Data Integrity</b>	
Data Integrity Constraints . . . . .	97
Single Table Constraints. . . . .	97
Multiple Table Constraints . . . . .	98
Referential Integrity . . . . .	98
Database Support for Constraints . . . . .	99
Stored Procedures & Triggers . . . . .	100
Stored Procedures . . . . .	100
Executing Stored Procedures . . . . .	101
Triggers . . . . .	102
<b>10 Transactions</b>	
The Transaction Mechanism . . . . .	103
Two-Phase COMMIT . . . . .	104
Data Replication . . . . .	104
Locking . . . . .	105
Isolation Levels . . . . .	106
Setting the Isolation Level . . . . .	107
Locking Levels . . . . .	107

Escalating a Lock to a Table Lock . . . . .	107
Enforcing Locks . . . . .	108
Locking Duration . . . . .	108

## **11 System Tables and Security**

System Tables . . . . .	109
How System Tables are Implemented . . . . .	110
Oracle Data Dictionary . . . . .	110
Informix, Sybase, MS-SQL, DB2 Data Dictionaries . . . . .	111
How to Use System Tables . . . . .	111
System-Stored Procedures in MS-SQL and Sybase . . . . .	112
Security . . . . .	113
System Security . . . . .	113
Data Security . . . . .	113
Using the Operating System User Name . . . . .	114
Oracle OPS ACCOUNT . . . . .	114
Informix . . . . .	115
Oracle/Rdb . . . . .	115
Granting and Revoking System Privileges . . . . .	115
System Privilege . . . . .	115
Object Privilege . . . . .	116

## **12 The Optimizer**

How the Optimizer Works . . . . .	117
The Access Path . . . . .	118
Update Statistics . . . . .	120
MS-SQL and Sybase . . . . .	120
Oracle . . . . .	120
Hinting the Optimizer . . . . .	121
Track the Optimizer Decision and Access Path . . . . .	121
MS-SQL and Sybase . . . . .	121

Oracle . . . . .	122
Informix . . . . .	124
Oracle/Rdb . . . . .	125
Examples . . . . .	125
SQL Command to Create the Data . . . . .	125
Simple SELECT Statement . . . . .	126
SELECT Statement with WHERE and ORDER BY Clauses . . . . .	127

### **13 Introduction to Magic & SQL**

Why Use SQL with Magic . . . . .	129
Magic SQL Support . . . . .	130
Built-in Support - The Implicit Level . . . . .	130
Embedded SQL - The Explicit Level . . . . .	131
Magic & SQL Terminology . . . . .	131
Magic Gateways Naming Conventions. . . . .	132
Magic's API Implementation and Versions . . . . .	132
Magic Repository and Capabilities. . . . .	134
Magic Repository and SQL Data Dictionaries . . . . .	134
Magic's Data Manipulation Capability . . . . .	135
Magic & SQL Features and Benefits . . . . .	136

### **14 Client/Server Architecture**

Magic Client/Server Architecture . . . . .	140
Communication Gateway . . . . .	140
Definition SQL Gateway . . . . .	141
Full SQL Gateway . . . . .	141
RDBMS (Open) Client/Server Architecture . . . . .	142
A Heterogeneous Client/Server Configuration . . . . .	143
When to Use What . . . . .	144

## **15 Defining the RDBMS Environment**

Setting the RDBMS Environment Variables . . . . .	145
Sybase 10/11 . . . . .	145
Oracle . . . . .	146
Informix . . . . .	146
MS-SQL . . . . .	146
DB2 . . . . .	146
Connecting to the Database . . . . .	147
Sybase . . . . .	147
Oracle . . . . .	148
Informix . . . . .	149
MS-SQL . . . . .	150
Giving Database Permission . . . . .	150
Using Windows Icons . . . . .	151

## **16 Configure and Define the Magic Environment**

The MagicGate Database Gateway for SQL . . . . .	153
Configuring the MagicGate Database Gateway for SQL . . . . .	153
The Magic Gateways Naming Convention . . . . .	154
The Magic Gateway Version Convention . . . . .	155
The [MAGIC_ENV] Flags Section. . . . .	156
ISAM Transactions . . . . .	156
Display Full Messages . . . . .	157
Multi-user . . . . .	157
The [MAGIC_DBMS] DBMS Section . . . . .	158
Nulls . . . . .	158
Default Float Picture . . . . .	158
DBMS Properties . . . . .	159
DBMS Parameters . . . . .	159
The [MAGIC_DATABASES] Settings/Databases Section . . . . .	160



Multiple Databases . . . . .	166
Multiple Connections . . . . .	166

## **17 The Magic Repository**

Table Repository . . . . .	167
Name . . . . .	167
DB Table. . . . .	168
Database Name . . . . .	168
Table Properties . . . . .	170
Database Information for SQL Databases . . . . .	170
Table Properties . . . . .	174
Column Definition and Properties . . . . .	175
Index Definition and Properties . . . . .	180
Properties . . . . .	181
Get Definition . . . . .	183
Notes on Get Definition . . . . .	187
View Definition Loading . . . . .	188
Table Modification . . . . .	188

## **18 Mapping Data Types**

Magic Default Mapping . . . . .	191
Get Table Definition Mapping . . . . .	192
Enforcing a Data Type - SQL TYPE . . . . .	193
Magic Data Types . . . . .	193
Date and Time Mapping . . . . .	195
Defining Date Columns in Magic . . . . .	195
Mapping on Existing Data. . . . .	196
TIME Attribute . . . . .	197
Magic Equivalents for the RDBMS Data Types . . . . .	197
Informix Data Types . . . . .	198
Sybase Data Types . . . . .	199

Oracle Data Types . . . . .	200
SQL Server Data Types . . . . .	201
DB2 Data Types . . . . .	202
ODBC Data Types . . . . .	203
Data Definition Rules. . . . .	204

## **19 Understanding Magic & SQL Behavior**

Unique Identifier . . . . .	207
Unique Identifiers in RDBMSs . . . . .	207
Unique Identifier in Magic . . . . .	208
Position Index . . . . .	209
Unique and Non-Unique Indexes. . . . .	209
A Simple Magic Program . . . . .	210
Regular Link Operation . . . . .	211
Link Join Operation. . . . .	212
The Join Operation . . . . .	212
Join Operation Parameters. . . . .	213
Join Operation Usage . . . . .	214
Join Operation Behavior. . . . .	214
Ranging . . . . .	218
CNDRANGE( ) Function . . . . .	218
SQL Where Clause . . . . .	219
SQL Where Usage. . . . .	219
SQL Where Behavior . . . . .	221
SQL Where Usage Considerations . . . . .	222
One-Way and Two-Way Behavior . . . . .	223
Incremental Locate . . . . .	225
Sort Using RDBMS . . . . .	225
Sort Usage . . . . .	225

Sort Behavior . . . . .	226
Sort Usage Considerations . . . . .	226

## **20 Debugging Your Programs**

Magic Gateway Log . . . . .	227
How to Set the Flags . . . . .	227
The Flags . . . . .	228
Where to Set the Flags . . . . .	229
How to Work with the Log File . . . . .	229
RDBMS Tools . . . . .	229
Magic Profiler . . . . .	230
Flow Monitor . . . . .	230

## **21 Locking and Transaction Processing**

The Multi-user Flag . . . . .	232
Table Access and Share Mode . . . . .	232
Magic Locking Strategies . . . . .	233
Immediate . . . . .	233
On Modify . . . . .	234
After Modify . . . . .	234
Before Update . . . . .	234
No Lock . . . . .	235
Transaction Levels . . . . .	236
Task Level . . . . .	236
Record Level. . . . .	236
Transaction Level Rules . . . . .	239
Error Recovery Control Options . . . . .	239
On Record Locked Parameter . . . . .	240
Transaction Error Parameter. . . . .	240
LCK Parameter - Call Task or Call Program . . . . .	241
Physical and Logical Locks . . . . .	241

Physical Locks . . . . .	241
Logical Locks . . . . .	243
BLOBs Locking . . . . .	244
Isolation Levels and Optimizer Hints . . . . .	244
The ROLLBACK Function . . . . .	244
The Intrans Function . . . . .	245

## **22 Explicit SQL**

Using Explicit SQL . . . . .	248
Open the Task menu . . . . .	248
Explicit Embedded SQL Elements . . . . .	250
Create an Embedded SQL Task . . . . .	250
The SQL Command . . . . .	250
Input Parameters . . . . .	251
Output Parameters . . . . .	251
The Assist Utility . . . . .	251
The SQL Command Automatic Program Generator - APG . . . . .	253
Behavior of Explicit SELECT Statements . . . . .	255
Online vs. Batch . . . . .	255
Result Database as Input Database . . . . .	255
Result Database Different from Input Database . . . . .	256
Recommendations . . . . .	256
Restrictions on Using Embedded SQL . . . . .	257
Error Handling . . . . .	258
The DBERR Function . . . . .	258

## **23 Magic & SQL Performance**

Key Definition and Usage . . . . .	259
Range Definition . . . . .	262
Transactions . . . . .	263
Online Tasks . . . . .	263

Batch Tasks . . . . .	263
Sorting . . . . .	264
Embedded or Explicit SQL . . . . .	264
Views . . . . .	265
Stored Procedures and Triggers . . . . .	266
Client/Server . . . . .	267

## **24 Magic 8 Migration and Portability**

What's New in Magic 8 . . . . .	269
Migrating from Magic 6 or 7 to Magic 8 . . . . .	270
ISAM – RDBMS . . . . .	271
RDBMS – ISAM . . . . .	274
RDBMS – RDBMS . . . . .	274
Keeping the Lowest Common Denominator . . . . .	275

## **25 Technical Information**

Magic SQL Command Object . . . . .	277
General. . . . .	277
Select Statements . . . . .	277
Stored Procedure. . . . .	278
Result Database . . . . .	278
Automatic Program Generator. . . . .	278
Other Statements. . . . .	279
General Issues . . . . .	279
SQL DATETOALPHA Parameter . . . . .	279
Security . . . . .	279
Error Handling. . . . .	280
Sort/Temporary Database . . . . .	280
Triggers . . . . .	280
Rules. . . . .	280

Oracle . . . . .	281
Optimizer Hints . . . . .	281
Stored Procedure. . . . .	281
Multi-Connections. . . . .	282
Views . . . . .	282
Unique Views . . . . .	282
Two Phase Commit . . . . .	282
Create Table Parameters. . . . .	283
Load Definition from a Remote Server . . . . .	283
Deferred Execution of SQL Steps . . . . .	284
NLSSORT Support . . . . .	284
MS-SQL . . . . .	285
Optimizer Hints . . . . .	285
Multi-Connections. . . . .	286
Views . . . . .	286
Locking . . . . .	286
DB Commands. . . . .	286
Relevant Parameters . . . . .	287
Sybase . . . . .	289
Optimizer Hints . . . . .	289
Locking . . . . .	290
Views . . . . .	290
Multiple Connections . . . . .	290
ODBC. . . . .	291
Locking . . . . .	291
Tested ODBC Drivers . . . . .	291
Views . . . . .	292
Check Driver Program. . . . .	292
Default Values. . . . .	292
Known Problems . . . . .	292

The ODBC Check Driver Utility: MGCHCKDRV.EXE . . . . .	293
ODBC Gateway - Data Source Information. . . . .	294
DB2. . . . .	317
Timeout Locks. . . . .	318
Views . . . . .	318
Using DB2 Handles . . . . .	318
Informix. . . . .	320
Views and Fragmented Tables. . . . .	320
Locking . . . . .	320
Multiple Connections . . . . .	321
Text and Byte Data Types . . . . .	321
Compatibility with Previous Versions. . . . .	322
Properties Supported by SQL Gateways . . . . .	325

**A Basic SQL Command Syntax**

**B Example Database Information**

Products . . . . .	331
Suppliers . . . . .	331
Inventory . . . . .	332
Offices . . . . .	332
Departments . . . . .	332
Employees . . . . .	333

<b>Index</b> . . . . .	335
------------------------	-----

[This page intentionally left blank]



# Introduction and Overview

# 1

---

**T**he *Magic Guide to SQL* shows you how to implement Magic with your SQL relational database application. The guide also discusses the Magic & SQL interface, and provides in-depth technical information about how to configure your system and optimize performance.

This guide is aimed primarily at Magic users who want to develop, support, and maintain portable and highly interoperable SQL database applications. Such applications will be less cumbersome than the current offerings of 3GLs, 4GLs, and SQL fourth generation end tools.

Because Magic is such a powerful and flexible program, it can be used with many different SQL databases. The interface between Magic and each database varies slightly. *The Magic Guide to SQL* points out the differences among five popular SQL database programs: DB2, Oracle, Oracle/Rdb, Sybase, and Informix. Read the chapters relevant to your database environment to understand exactly how the interface between Magic and your database environment works.

Magic is an open systems product. This means that Magic can be used to develop applications that will be deployed across a variety of operating systems, a variety of configurations, and a variety of File Management and Relational Database Management systems.

## **Audience**

*The Magic Guide to SQL* is a technical guide intended for experienced Magicians who have completed both the Level 1 and Level 2 Magic University courses, or who have equivalent experience with Magic.

You should have the appropriate SQL DBMS installed and operational on your system. You should also have at least a minimal understanding of SQL and DBMSs.

For additional training in Magic & SQL, consult your local MSE distributor for information on Magic University courses in your area.

## **Scope**

*The Magic Guide to SQL* consists of two parts.

The first part, from Chapter 1 through Chapter 12, will familiarize you with SQL language, features, and techniques, including the following topics:

- First introduction to SQL
- Using SELECT statements
- Database structure
- Updating data
- Transactions
- RDBMS mechanisms: security, optimizer

The second part, from Chapter 13 through Chapter 25, analyzes the interface between Magic & SQL, and includes the following topics:

- Performance advantages and disadvantages of using SQL for a file manager
- How Magic addresses problems that may arise

## ***Objectives of the Guide***

With *The Magic Guide to SQL at your side*, you should be able to:

- Understand basic SQL Data Manipulation Language (DML) and Data Definition Language (DDL) statements
- Convert an existing ISAM Magic application to work with an SQL DBMS
- Understand the performance advantages and disadvantages of using SQL for the file manager
- Mix appropriate file managers to achieve desired performance
- Understand basic optimizer working rules
- Feel at home with known database access tools

## ***Examples***

The examples used throughout this guide are based on data that can be found in Appendix B, Example Database Information.

# ***SQL Structure and Features***

SQL stands for Structured Query Language. SQL is used in different physical files to organize, manage, and retrieve data in computer databases. SQL interacts with a specific type of database, called a relational database. In a relational database information is organized into related tables, as opposed to a flat-file database where data is stored in files that are not related to each other.

The computer program that controls the database is called a database management system, or DBMS. When the database uses a relational architecture, it is called a relational database management system, or RDBMS.

## ***The SQL Engine***

SQL isolates the end-user and the application programs from the database. When you need to retrieve data from the database, you use SQL to make the request. All attempts to update, change, or define the database are intercepted by the SQL engine. It is the engine's responsibility to validate the request and then to apply the request to the database. This allows the SQL engine to both verify and optimize the request.

The RDBMS processes the SQL request, retrieves the requested data, and returns it to the requester. This process is called a database query, and is the source of the name Structured Query Language. However, SQL is more than a query tool because it is also used to control all the functions that the RDBMS provides for its users.

## ***DBMS Functions Controlled by SQL***

### ***Data Definition***

SQL lets the user define the structure and organization of the data, as well as the relationships among the data elements in the database.

### ***Data Retrieval***

SQL lets the user of an application program retrieve stored data.

### ***Data Manipulation***

SQL lets the user of an application program update a database by adding new data, deleting old data, and modifying existing data.

### ***Access Control***

SQL can be used to restrict user access to data. A user can have either view-only or modify rights.

### ***Data Sharing***

By definition, an SQL database is a multi-user system that allows concurrent access to the database.

### ***Data Integrity Control***

SQL lets the developer define integrity constraints, which are stored in the database. Therefore, no matter how the data is manipulated, either via an interactive SQL session or via any application program, the SQL database protects the data from corruption.

Data integrity is also preserved by using transactions, which are implemented by transaction control statements such as SET TRANSACTION, COMMIT, and ROLLBACK.

The SQL language consists of about thirty statements, summarized in the table below. Each statement requests a specific action from the DBMS, such

as creating a new table, retrieving data, or inserting new data into the database.

---

<b>SQL Statement</b>	<b>Description</b>
<b>Data Manipulation</b>	
SELECT	Retrieve data from the database
INSERT	Add new rows of data to the database
DELETE	Remove rows of data from the database
UPDATE	Modify existing data in the database
<b>Data Definition</b>	
CREATE TABLE	Add a new table to the database
DROP TABLE*	Remove a table from the database
ALTER TABLE*	Change the structure of an existing table
CREATE VIEW	Add a new view to the database
DROP VIEW*	Remove a view from the database
CREATE INDEX*	Build an index for a column
DROP INDEX*	Remove an index for a column
CREATE SYNONYM*	Define an alias for a table name
DROP SYNONYM*	Remove an alias for a table name
COMMENT*	Define remarks for a table or column
LABEL*	Define a title for a table or column
<b>Access Control</b>	
GRANT	Grant user access privileges
REVOKE	Remove user access privileges

---

SQL Statement	Description
Transaction Control	
COMMIT	End (commits) the current transaction
ROLLBACK	Abort the current transaction
Programmatic SQL	
DECLARE	Define a cursor for a query
EXPLAIN*	Describe the data access plan for a query
OPEN	Open a cursor to retrieve query results
FETCH	Retrieve a row of query results
CLOSE	Close a cursor
PREPARE*	Prepare an SQL statement for dynamic execution
EXECUTE*	Execute an SQL statement dynamically
DESCRIBE*	Describe a prepared query *

\* Not part of the ANSI/ISO SQL standard but found in many popular SQL-based products

All of the SQL statements described above have the same basic structure. Each statement consists of a verb and one or more clauses, and performs a single specific function.

# ***Basic Relational Database and SQL Concepts***

## ***Table***

A table is a named, two-dimensional representation of the data values. Each table in an RDBMS has a unique name and can be compared to a file in an ISAM file system. All of the data in a table is related to a specific topic and is organized into columns (fields) and rows (records).

## ***Row***

Each record in a table is called a row. The structure of information in each row of a single table is the same. A single SQL statement can work on one or multiple rows at the same time. This is very different from an ISAM file manager, which processes only one record (row) at a time.

## ***Column***

Each row contains one or more columns. In an ISAM file manager columns are called fields. All of the data in one column in a table will have the same basic attribute and size. For example, a customer table could have one column for the customer number (a 6-digit numeric field), another column for the first name (30-character alpha field), etc.

## ***Null Value***

Nulls represent missing and unknown data. All SQL databases support null values. A field that has a null value is different from a blank field in an ISAM file. Null means that the value is not known. Null values require special handling. If you attempt to do arithmetic on a numeric column and one or more of the values are null, then the result will be null. If an alpha field allows null values, and you select all records in which the alpha field is blank, records with the null value in the alpha field will NOT be selected.



## ***View***

A view is a virtual database object. Physically the data is stored in an internal structure, but the data is logically organized into tables for viewing ease.

Views enable an end-user to view data in one or more tables through a window. A view is defined by a SELECT statement and can be thought of as a named, stored, SELECT statement. However, views are usually thought of as tables without actual data. Views are often treated as tables and have column names created from the SELECT statements that define them.

## ***Cursor***

A cursor is a symbolic name associated with an SQL statement that is called the body of the cursor. When a cursor is opened, the body of the cursor is executed, generating a result set. Cursors let applications retrieve data by fetching the result set rows one by one.

# ***Elements of Database Application Development***

All database application development techniques require two common elements, regardless of whether the development tool is language-based or non-language-based, procedural or non-procedural, and whether the tool is object-oriented or not:

- Data Definition Language - DDL. To define and describe data.
- Data Manipulation Language - DML. To specify selective data retrieval, either in groups or as individual records, combined with the ability to manipulate the retrieved data.

## ***SQL Data Definition Language***

The DDL is the set of operations that define and alter the database table structure, such as the CREATE TABLE and ALTER TABLE commands. DDL is also used for structural enhancements such as CREATE VIEW, which creates a window to the actual tables.

## ***SQL Data Manipulation Language***

DML is the set of operations used to change data. Where DDLs define and modify the table structure, DMLs work on the data stored in the tables. Use DMLs to either query or update data in the database. In SQL, DML statements have a consistent and relatively simple syntax as described below.

Note: Throughout this guide, variables in examples are displayed in italics within parentheses. When entering an SQL statement, replace the variable with an actual value.

### ***Querying the database***

Data retrieved from the database is returned as a virtual table, formatted row by row, and ordered as specified by the query.

- **SELECT** (*column list*) - Lists the data items to be retrieved by the SELECT statement.
- **FROM** (*table list*) - Lists tables that contain the data to be retrieved by the query.
- **WHERE** (*condition list*) - Specifies which data to select in response to the query.
- **GROUP BY** (*column list*) - Specifies a summary query, which groups similar rows, producing one row for each group.
- **HAVING** (*condition list*) - Requests SQL to include only certain groups produced by the GROUP BY clause in the query results.
- **ORDER BY** (*column list*) - Sorts the query results based on the data in one or more columns. If omitted, the query results are not sorted.

## ***Updating the database***

Modify data in a relational database one table at a time, using the following commands:

- INSERT INTO (*table*) - Adds one or more new rows (records) to a table.
- UPDATE (*table*) - Names the table to be modified.
- SET (*column\_set\_statements*) - Names the columns to be updated and how to update them.
- WHERE (*condition list*) - Limits the rows (records) to be modified.
- DELETE FROM (*table*) - Deletes one or more rows (records) from table.
- WHERE (*condition list*) - Limits the rows (records) to be deleted.

## ***The SQL Database Administrator***

The database administrator (DBA) is a person responsible for placing tables on physical disks, backing up and recovering files, and granting system access. It is also the database administrator's job to optimize the underlying physical representation of the data according to the organization's needs.

Another way to optimize a database is to use indexes. All access to RDBMS tables can be done without indexes. However, the use of indexes can enhance performance. Most developers create indexes when they create tables, but optimizing an index usually takes place during the tuning process and usually involves working with the DBA.

The DBA is usually highly skilled in the RDBMS and is the only one authorized to modify the database design. This is different from the ISAM system manager, who can be any developer or programmer who can create and alter the file structure.

## ***Magic University International - MUI***

Magic Software Enterprises is dedicated to providing the latest and most efficient training so that you may increase your productivity and stay ahead in your work environment. Magic University International courses offer you learning opportunities that cannot be provided by a book alone.

The Magic University International course on Magic & SQL will provide you with:

- Hands-on training
- Classroom practice of various concepts and facilities, such as:
  - multi-user applications
  - locking strategies
  - the SQL Gateway log
- Up-to-date information on:
  - specific Magic gateways
  - specific DBMS versions and support
- A high-level instructor
- Tips and tricks for working with the most recent versions of your particular RDBMS and of Magic & SQL

For more information on MUI courses in Magic & SQL, contact the MSE distributor in your area.

---

**T**he ANSI/ISO SQL standard specifies various types of data that can be stored in an SQL-based database and manipulated by the SQL language. The data types specified by the standard are only a minimal set, but almost all commercial SQL products support them or have data types that are very similar to them. The ANSI/ISO data types are summarized below:

## *ANSI/ISO Data Types*

- **Fixed-length character strings** - Used to store text elements, such as individual and company names, addresses, descriptions.
- **Integers** - Used to store counts, quantities, ages. Integer columns can contain id numbers, such as customer, employee, and order numbers.
- **Decimal numbers** - Used to store numbers that have fractional parts and must be calculated exactly, such as rates, percentages, and currency values.
- **Floating-point numbers** - Used to store scientific numbers that can be calculated approximately, such as weights and distances. Floating-point numbers can represent a larger range of values than decimal numbers, but can produce small rounding errors in computations.

---

<b>Data Type</b>	<b>Description</b>
<b>CHAR</b> ( <i>len</i> )	Fixed-length character string
<b>CHARACTER</b> ( <i>len</i> )	
<b>INTEGER</b>	Integer number
<b>INT</b>	
<b>SMALLINT</b>	Small integer number
<b>NUMERIC</b> ( <i>precision, scale</i> )	Decimal number
<b>DECIMAL</b> ( <i>precision, scale</i> )	
<b>DEC</b> ( <i>precision, scale</i> )	
<b>FLOAT</b> ( <i>precision</i> )	Floating point number
<b>REAL</b>	Low-precision floating point number
<b>DOUBLE PRECISION</b>	High-precision floating point number

---

## ***Extended Data Types***

Most commercial SQL products offer a more extensive set of data types than the ANSI/ISO standard. Some extended data types are:

- **Variable length character string** - Almost every SQL product supports VARCHAR data, which allows a column to store character strings that vary in length from row to row, up to some maximum length. The ANSI/ISO standard specifies only fixed-length strings, which are padded on the right with trailing blanks.
- **Currency** - Many SQL products support a MONEY or CURRENCY type, which is usually stored as a decimal or floating-point number. Having a distinct money type allows the DBMS to properly format money amounts when they are displayed.
- **Date and Time** - Support for date and time values is also common in SQL products, although the details vary from one product to another. Various combinations of dates, times, time stamps, time intervals, and date/time arithmetic are generally supported.
- **Boolean data** - Some SQL products, including dBASE IV, support logical TRUE, FALSE values as an explicit type, such as BIT in SYBASE.
- **Long text** - Several SQL-based databases support columns that store long text strings, typically up to two gigabytes. The DBMS usually restricts the use of these columns in interactive queries and searches.
- **Unstructured byte streams** - Oracle and several other products allow unstructured variable-length sequences of bytes to be stored and retrieved. Columns containing this data type are used to store compressed video images, executable code, and other types of unstructured data, such as IMAGE in Microsoft SQL and LONGDRAW in ORACLE.
- **Asian characters** - DB2 supports fixed-length and variable-length strings of 16-bit characters used to represent Kanji and other Asian characters. However, searching and sorting in these graphic types are not permitted.

See Appendix A for a table that summarizes the data types supported by several popular SQL-based database products.

## ***Data Type Limitations***

Each data type has its limitations.

All SQL products allow only valid dates to be inserted.

### **Examples:**

- Oracle - DATE
- Sybase/MS-SQL - DATETIME

Note that a zero date is not a valid value.

Most DATE data types actually include a time portion.

## ***Data Type Differences***

Differences among data types offered in various SQL implementations create obstacles to the portability of SQL-based applications. Subtle differences in data types among various SQL products lead to some significant differences in SQL statement syntax. These differences can even cause the same SQL query to produce different results on different database management systems.

The SQL claim of portability is true, but only at a general level. An application can be moved from one SQL database to another, but the subtle variations in SQL implementations mean that data types and SQL statements must be adjusted in the process. Transparent portability of SQL-based applications remains a goal but is not a reality.



## Constants

Constants are specific values used in SQL statements. A constant can be the value you want to insert in a column of a row, or a value you want to use in comparison with a column, or a value to select as part of the SELECT statement. Constants can be numeric, character, or date, and can be used in several SQL contexts.

You can use a constant anywhere you might want to compare or insert values. Character and date constants must be enclosed in single quotes to differentiate them from column names and reserved words. Number constants can be used without quotes because column and table names always begin with a character.

```
INSERT INTO SALESREP (EMPL_NUM, NAME, QUOTA,
HIRE_DATE, SALES)
VALUES (115, 'Dennis Irving', 175000.00,
'21-JUN-90', 0.00)
```

The value for each column in the newly inserted row is specified in the VALUES clause. Constant data values are also used in expressions, such as in this SELECT statement:

```
SELECT CITY
FROM OFFICES
WHERE TARGET (1.1 * SALES) + 10000.00
```

The ANSI/ISO SQL standard specifies the format of numeric and string constants, or literals, which represent specific data values. These conventions are followed by most SQL implementations.

### ***Numeric Constants***

Integer and decimal constants (also called exact numeric literals) are written as ordinary decimal numbers in SQL statements, with an optional leading plus or minus sign.

## Common RDBMS Data Types

Data Type <sup>1</sup>	DB2 and SQL/DS	Oracle <sup>2</sup>	Informix	Sybase and MS-SQL Server
Fixed-Length Character	CHARACTER(n)	CHAR(n)	CHAR(n)	CHAR(n)
Variable-Length Character	VARCHAR(n)	VARCHAR2(n) n up to 2000	VARCHAR(n) n<256	VARCHAR(n)
Long Text	LONG VARCHAR	LONG	TEXT	TEXT
Integer	SMALLINT INTEGER	NUMBER(p)	SMALLINT INTEGER	TINYINT SMALLINT INT
Decimal	Decimals (p,s)	NUMBERS(p,s)	Decimals (p,s)	
Money			MONEY(p,s)	MONEY
Floating Point	FLOAT(p) REAL DOUBLE PRECISION		SMALLFLOAT FLOAT	FLOAT
Date/Time	DATE TIME TIMESTAMP	DATE <sup>3</sup>	DATE DATETIME TO... INTERVAL TO...	DATETIME
Boolean				BIT BINARY(n) VARBINARY(n) IMAGE
Byte Stream		RAW LONG RAW		SYSNAME USER_TYPE_NAME

Data Type <sup>1</sup>	DB2 and SQL/DS	Oracle <sup>2</sup>	Informix	Sybase and MS-SQL Server
Other	GRAPHIC(n) VARGRAPHIC(n) LONG VARGRAPHIC		SERIAL	

<sup>1</sup> Many of the products also support the ANSI/ISO types as synonyms to their own types.

<sup>2</sup> Oracle stores all numbers using its own internal format with precision to 40 positions.

<sup>3</sup> The Oracle DATE data type is really a timestamp. Oracle built-in functions can be used to isolate the date and time parts.

## Summary

The ANSI/ISO standards provide several data types: Character, Integer, SmallInt, Numeric, Decimal, Float, Real, and Double Precision. However, the databases commonly support extended data definitions, such as Varchar, Money, Date, Boolean, Long, and Raw. These extensions are necessary because of user needs but reduce portability among database types.

[This page intentionally left blank]

# Connecting to a Database

---

# 3

**A**n RDBMS is made up of servers and databases. We connect to the database from the user interface using the *connect string*.

## SQL Servers and Databases

The SQL server, which receives and processes the SQL commands, runs the RDBMS software and handles the functions required for concurrent shared data access.

The SQL server is the software that manages the databases. A client is an application that requests information from a server. Every time a server is started, a memory area is allocated and the server background processes are initiated.

Most database systems support many databases on one server.

When working with Oracle, the terminology is a bit different. One Oracle server can have one or more instances, which are equivalent to databases in other SQL servers. Each instance can include only one database. One or more instance can be on each server machine. Each server instance is identified by its SID.

## **SQL User Interface**

Each RDBMS has an interactive user interface. After connecting to the user interface, the user can execute any SQL statement available in that particular RDBMS. The interactive user interface gives the user the opportunity to execute SQL commands and SQL script files, to create stored procedures, etc.

The interactive user interfaces are:

- Sybase - ISQL (DOS,UNIX), WISQL (Windows)
- MS-SQL - ISQL/W
- Oracle - SQL/PLUS
- Informix- dbaccess
- DB/2 - DB2 command line
- Oracle/Rdb - SQL

In addition to the interactive user interface, the user can execute SQL commands through third generation languages such as C.

## ***Connect String***

To work with the SQL database and execute SQL commands the user must first connect to the SQL server by sending a connect string to the database. The connect string includes the user name and password for the user wanting to connect, and a parameter for the database server. (Under Oracle, the SID identifies the database server to be connected.)

Each database has a different connect string structure.

When working with a remote Oracle server, in addition to the user name and password, we need to add an SQL\*NET1 connect string that includes the network protocol, the Oracle server host name, and the instance SID.

### **Examples:**

- Sybase: ISQL U username  
P password  
S servername
- Oracle: (local)  
sqlplus username/password  
(remote)  
sqlplus username/password@SQL\*NET  
connect-string

Once the connection has been made to an SQL server, the user may choose which database to use.

Most RDBMSs use a GUI interface for connection. However, the connect command line is also used in many cases.

## ***SQL Reconnect***

If the connection between the Magic engine and the SQL server is suddenly disrupted, Magic automatically reconnects to the SQL server when there is no open cursor.

[This page intentionally left blank]



# *Creating Database Objects*

# 4

**I**n many cases SQL users do not have to worry about creating a database. Instead they use interactive or embedded SQL to access corporate or other databases.

However there are times when you need to create private tables for personal data, such as sales forecasts. If you use a multi-user database, you may want to create tables or databases that will be shared with other users. If you use a personal computer database, you will want to create your own tables and databases to support your personal applications.

## *Creating a Database*

A database contains all the parts that go into a data model. In addition to tables, these parts include views, indexes, and other objects associated with the database. You must create a database before you can create anything else.

The database cannot be manipulated by the operating system. When you create a database, the RDBMS sets up records that show the existence of the database. These records are not visible to the operating system because the RDBMS manages disk space directly.

The physical location of objects is transparent, but you can specify a physical location and special parameters for objects. You can also have your database span across different locations. Terms used to describe physical locations include: tablespace, dbspace, segments, and devices.

Use the CREATE DATABASE command to specify location and other relevant parameters. For example:

- Informix

```
CREATE DATABASE dbname WITH BUFFERED LOG
```

- Sybase and MS-SQL

```
CREATE DATABASE dbname ON device1=16 LOG ON  
logdev 1=10
```

## ***Altering Database Parameters***

After a database has been created, you may need to alter it and change the SQL server parameters that affect the database. Use the ALTER DATABASE command to expand or change your database.

## ***The Data Definition Language - DDL***

A set of SQL statements called Data Definition Language (DDL) handles structural changes to a database.

Using DDL statements you can:

- Define and create a new table or view
- Rename an existing table or view
- Remove a table or view
- Change the definition of an existing table or view
- Establish security controls for a database
- Build or remove an index
- Control the physical storage of data by the DBMS

DDL is based on four SQL actions: CREATE, DROP, ALTER, and RENAME.

- CREATE - defines and creates a database object

**Syntax:**

```
CREATE [TABLE,INDEX] name (column definitions)
```

The column definitions are unique-column-name data-type.

Use the NOT NULL option after the data type to restrict null entry. NOT NULL is the default unless otherwise specified.

**Examples:**

```
CREATE TABLE custs (custno INTEGER,  
custnameCHAR(30), hire_date DATETIME)
```

When you create a table you need to give the table a name and describe the columns the table will contain, including:

- column name
- type - character, date, number
- length

Some data types are database-specific. Nevertheless, ANSI-standard data types are found in many DBMSs.

In some RDBMSs you can create a new table from the results of a SELECT statement. The fields of the new table created from the results of a SELECT statement have the same attributes as in the selected table.

```
CREATE [UNIQUE] [CLUSTERED] INDEX index_name
ON table name (column name[,column name])
```

When you create an index you can define the index to be clustered. The physical data in the table will then be placed according to the order in the index. This feature is especially helpful when you have a unique index that is used often.

- DROP - removes an existing database object

**Syntax:**

```
DROP TABLE (tablename)
```

You may want to delete a table that is no longer useful or to rebuild a table during development. Execute the DROP TABLE command to delete the table and all related indexes.

- ALTER - changes the definition of a database object

**Syntax:**

```
ALTER TABLE (tablename) ADD columnname
column-definition
```

Sometimes you need to change tables in an application or file. You can use a single ALTER TABLE command to make many of these changes. There are two general types of changes:

- Add a column
- Modify an existing column

- RENAME - renames an object name

**Syntax:**

```
RENAME TABLE old tablename new tablename
```

Sometimes you need to rename an object in the database. The object may be a table, column, stored procedure trigger, etc.

Syntax varies among the different RDBMSs. For example, in SYBASE and Microsoft SQL a stored procedure, `sp_rename`, is used to rename an object name instead of the RENAME command.

Note: In most implementations DDL statements lock the data dictionary but only change the definitions of the tables and not the data rows themselves.

## Views

When you create tables in a database you may also want to create views to help you access those tables. Any DML statement that uses a table name can use a view name instead. Views are used to implement security, to simplify DML statements, and to improve performance.

There are some restrictions on using views. You cannot insert into, update, or delete from views that are defined for more than one table. You also cannot update views created from SELECT statements that have expressions other than `column_names` in the select list.

Note: Views do not contain the data from the SELECT statement. When used in place of a table name, the view's SELECT statement is concatenated to the SELECT statement and executed together.

[This page intentionally left blank]

# Selecting and Sorting Columns

# 5

The **SELECT** statement retrieves data from a database and returns it to you as a table. The columns specified in the **SELECT** statement are the names of the database columns that contain the data you want to retrieve. The database you are querying, sometimes called the base table, is specified in the **FROM** clause.

The basic elements of a **SELECT** statement are:

```
SELECT FROM [WHERE] [GROUP BY] [HAVING] [ORDER BY]
```

The minimum form of a **SELECT** statement is:

```
SELECT FROM
```

## Selecting All the Columns From a Table

You can retrieve all the columns from a base table in any order you want, regardless of the column order in the base table.

There are two different ways to select all the columns of a base table:

- Enter all the column names, separated by commas, and the name of the base table where those columns are defined. The order of the columns in the resulting table will be the same as the order specified in the query.

- Enter an asterisk and the name of the base table where the columns are defined. The order of the columns in the resulting table will be the same as the column order in the base table.

**Syntax:**

```
SELECT column_name[, column_name . . .]  
FROM table_name
```

or

```
SELECT *  
FROM table_name
```

**Examples:**

List all the information about the departments.

1. Explicitly list the column names in the SELECT statement:

```
SELECT dname, location, deptno FROM departments
```

All the information from the Departments table is returned in a table. The order of the columns in the table is the order specified in the query.

Dname	Location	Deptno
Accounting	Los Angeles	10
Marketing	San Francisco	20
Operations	Norfolk	30
Sales	New York	40
Research	Berkeley	50



2. Include an asterisk in the SELECT clause.

```
SELECT *  
FROM departments
```

All the information from the Departments table is returned in a table. The order of the columns is the same as in the base table.

Deptno	Dname	Location
10	Accounting	Los Angeles
20	Marketing	San Francisco
30	Operations	Norfolk
40	Sales	New York
50	Research	Berkeley

## Selecting Some Columns From a Table

You can use the SQL SELECT statement to retrieve data from a subset of columns in the base table .

### Syntax:

```
SELECT column_name[, column_name. . .]  
FROM table_name
```

### Example:

List the employees and their jobs.

```
SELECT ename, job  
FROM employees
```

The resulting table lists each employee name and the corresponding job. Note that the base table includes additional columns that were not selected.

Ename	Job
Allen	Salesman
George	Salesman
Turnbull	Salesman
Markson	Clerk
Jordan	Analyst
Walker	President
Major	Manager
Klein	Analyst
Barton	Manager
Eiden	Manager
Anton	Clerk
Carlisle	Clerk
Bonfiglio	Analyst
O'Neil	Manager

## How to Eliminate Duplicate Rows

When you query only a few columns in a table, duplicate rows may be returned because a row is returned for each row in the table.

### Example:

List all the jobs in the company.

```
SELECT job
FROM employees
```

The resulting table lists the job entry for each employee entry in the Employees table.

---

Job
Salesman
Salesman
Salesman
Clerk
Analyst
President
Manager
Analyst
Manager
Manager
Clerk
Clerk
Analyst
Manager

---

You can use the DISTINCT keyword at the beginning of the SELECT statement to eliminate duplicate rows in query results.

**Syntax:**

```
SELECT DISTINCT column_name[, column_name . . .]  
FROM table_name
```

**Example :**

List all the jobs in the company.

```
SELECT DISTINCT job  
FROM employees
```

The result table lists only unique entries.

<hr/> Job <hr/>
Analyst
Clerk
Manager
President
Salesman <hr/>

## Sorting Query Results - the ORDER BY Clause

The ORDER BY clause is included in the SELECT statement when you want the rows of a query result to be returned in a specific order. If you do not specify the ORDER BY clause, the rows will not be returned in any particular order.

```
SELECT FROM [WHERE] [GROUP BY] [HAVING] [ORDER BY]
```

The ORDER BY clause consists of the keywords ORDER BY, followed by a list of order specifications separated by commas. An order specification can be either a column\_name or a number that specifies an expression in the select list.

### Syntax:

```
SELECT column_name[, column_name . . .]  
FROM table_name  
ORDER BY column_name
```

### Example:

List the sales for each office sorted by region, and within each region by city, in alphabetical order.

SELECT the columns according to the order you want them to have in the query results. Then specify which table the data is taken FROM and the first column you want to ORDER BY. When you want internal order, add the relevant columns.

```
SELECT city, region, sales  
FROM offices  
ORDER BY region, city
```

**Result:**

---

City	Region	Sales
Atlanta	Eastern	367,911
Chicago	Eastern	735,042
New York	Eastern	692,637
Denver	Western	186,042
Los Angeles	Western	835,915

---

By default SQL sorts data in ascending order. To specify descending order, use the following command:

**Syntax:**

```
SELECT column_name[, column_name . . .]  
FROM table_name ORDER BY column_name  
[DESC][column_name [DESC] . . .]
```

**Example:**

List the sales for each office, sorted in descending order by sales:

```
SELECT city, region, sales  
FROM offices  
ORDER BY sales DESC
```

**Result:**

City	Region	Sales
Los Angeles	Western	835,915
Chicago	Eastern	735,042
New York	Eastern	692,637
Atlanta	Eastern	367,911
Denver	Western	186,042

**Example:**

In this example the values from two columns in the base table, sal and comm, are added together to produce a single column in the result table, sal+comm.

List the total salary plus commission for each employee, sorted in descending order by the total salary, which will be the second column in the result table:

```
SELECT ename, sal + comm
FROM employees
ORDER BY 2 DESC
```

**Result:**

Ename	Sal + Comm
Turnbull	9100
George	7400
Allen	6300
Walker	5500
O'Neil	4850
Major	4750
Eiden	4700
Jordan	4250
Barton	4200
Bonfiglio	4175
Klein	3900
Anton	3500
Carlisle	3375
Markson	3000

Some data types such as 'LONG'/TEXT cannot be sorted and therefore cannot appear in the ORDER BY clause.

You may also use the ORDER BY clause for columns that have not been selected.



# Row Selection - the WHERE Clause

# 6

**R**etrieving all of the rows from a base table is not always necessary. You will often want to limit the rows returned to those that meet specific criteria. The WHERE clause is used to limit the selection criteria.

## Selecting Rows with the WHERE Clause

The WHERE clause is the part of an SQL statement that limits the rows returned by the query.

```
SELECT FROM [WHERE] [GROUP BY] [HAVING] [ORDER BY]
```

The WHERE clause consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved. The search condition acts as a filter.

- Rows that satisfy the search condition pass through the filter and become part of the query results.
- Rows that do not satisfy the search condition are trapped by the filter and are excluded from the query results.

The WHERE clause is optional. If you do not include the WHERE clause in your SELECT statement, all the rows of the base table will be returned in the result. If the base table is large, the result table will be large as well.

```
SELECT name, sales
FROM salesreps
WHERE manager = 104
```

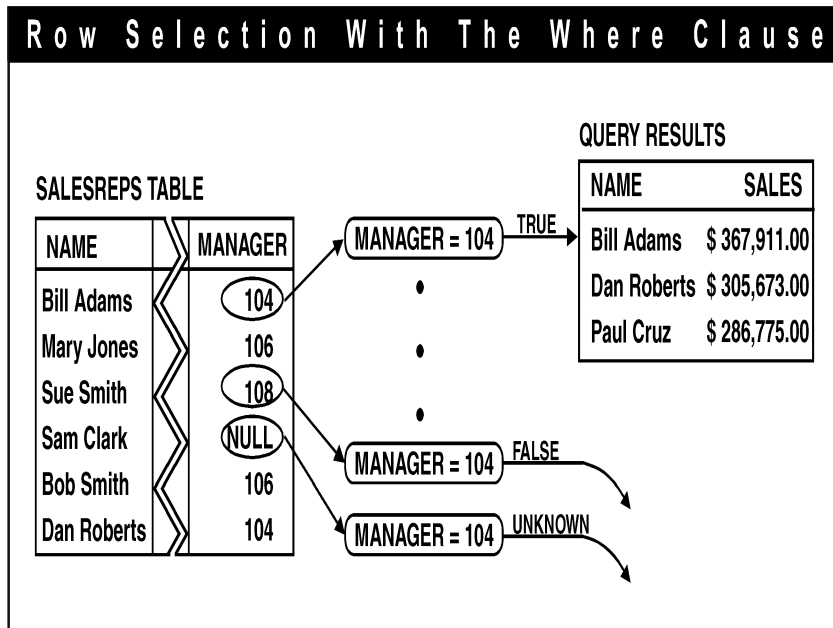


Figure 6-1 Row Selection with the Where Clause

## Search Conditions

SQL offers a wide variety of search conditions that let you specify many different kinds of queries efficiently. There are five basic search conditions:

- **Comparison test** (=, <>, <=, <, >, >=) - Compares the value of one expression to the value of another expression.

- **Range test (BETWEEN)** - Tests whether the value of an expression falls within a specified range of values.
- **Set membership test (IN)** - Checks whether the value of an expression matches one value in a set of values.
- **Pattern matching test (LIKE)** - Checks whether the value of a column containing string data matches a specified pattern.
- **Null value test (IS NULL)** - Checks whether a column has a NULL value.
- **Compound search conditions (AND, OR, and NOT)** - Combines search conditions with AND, OR, and NOT to form more complex search conditions.

**Syntax :**

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE condition

[condition,
condition . . .]
```

***Select Rows that Satisfy a Comparison Test***

Use a comparison test when you only want rows that meet the conditions of a comparison test.

- = Comparison Test

**Syntax:**

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name = expr
```

**Example:**

List the employee number, name, job, and salary for all employees in department 10.

```
SELECT empno, ename, job, sal
FROM employees
WHERE deptno = 10
```

**Result:**

Empno	Ename	Job	Sal
3123	Walker	President	5500
3115	Carlisle	Clerk	3375
3970	O'Neil	Manager	4850

- <> Comparison Test

**Syntax:**

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name <> expr
```

**Example:**

List the employee name, job, and salary for each employee who is not a manager:

```
SELECT ename, job, sal
FROM employees
WHERE job <> 'Manager'
```

**Result:**

Ename	Job	Sal
Allen	Salesman	2300
George	Salesman	2400
Turnbull	Salesman	2050
Markson	Clerk	3000
Jordan	Analyst	4250
Walker	President	5500
Klein	Analyst	3900
Anton	Clerk	3500
Carlisle	Clerk	3375
Bonfiglio	Analyst	4175

When comparing values to a string, the string is case-sensitive and must be enclosed in single quotes. If the condition in the above example had been entered as

```
WHERE job <> 'manager'
```

the result table would contain four Manager rows. MS-SQL can be case-sensitive.

## Select Rows that Satisfy a Range Test

Use a range test when you want only rows that have a value in a column within a specific range.

### Syntax:

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name BETWEEN expression AND
expression
```

### Example:

List the name, job, and hire date of each employee hired from March 1, 1985 to, and including, December 31, 1989. This example assumes that the conditional column is of an appropriate type.

```
SELECT ename, job, hiredate
FROM employees
WHERE hiredate BETWEEN '01-MAR-85' AND
'31-DEC-89'
```

### Result:

Ename	Job	Hiredate
Allen	Salesman	15 Apr 85
Jordan	Analyst	03 May 89
Walker	President	13 Mar 85
Major	Manager	13 Mar 85
Barton	Manager	02 Dec 87
Eiden	Manager	04 Aug 89

## Selecting Rows that Satisfy a Set Membership Test

Use a set membership test when you want only rows that contain a column value within a group or set of values.

### Syntax:

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name IN (value, value . . .)
```

### Example:

List the names of all employees who are salesmen or analysts.

```
SELECT ename, job
FROM employees
WHERE job IN ('Salesman', 'Analyst')
```

### Result:

Ename	Job
Allen	Sales
George	Sales
Turnbull	Sales
Jordan	Analyst
Klein	Analyst
Bonfiglio	Analyst

Use a compound search condition when you want only rows that contain a column value that is not within a group or set of values.

**Syntax:**

```
SELECT column_name[, column_name . . .]  
FROM table_name  
WHERE column_name NOT IN (value, value . . .)
```

**Example:**

List the names and jobs of all employees who are not salesmen or analysts.

```
SELECT ename, job  
FROM employees  
WHERE job NOT IN ('Salesman', 'Analyst')
```

**Result:**

ENAME	JOB
Markson	Clerk
Walker	Pres
Major	Mgr
Barton	Mgr
Eiden	Mgr
Anton	Clerk
Carlisle	Clerk
O'Neil	Mgr



## Select Rows that Satisfy a Pattern Matching Test

Use a pattern matching test when you do not know the exact value in a column, but you do know what you want from the column.

### Syntax:

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name LIKE pattern
```

A pattern is a string that includes wild card characters:

- The percent sign (%) wild card character can represent any number of characters, from 0 to *n*.
- The underscore (\_) wild card character can only represent one character. Each specific character in the pattern must be matched exactly.

### Example:

List the name, job, and department number for each employee whose last name begins with the letter B.

```
SELECT ename, job, deptno
FROM employees
WHERE ename LIKE 'B%'
```

### Result:

Ename	Job	Deptno
Barton	Manager	20
Bonfiglio	Analyst	50

**Example:**

List the name, job, and department number for each employee whose last name has six or more letters in it.

```
SELECT ename, job, deptno
FROM employees
WHERE ename LIKE '_ _ _ _ _ %'
```

The first five underscore marks ('\_') match the first five letters. The sixth underscore mark ('\_') matches at least one more letter. The final percent sign ('%') matches any other trailing characters.

**Result:**

Ename	Job	Deptno
George	Salesman	40
Turnbull	Salesman	40
Markson	Clerk	50
Jordan	Analyst	50
Walker	President	10
Barton	Manager	20
Carlisle	Clerk	10
Bonfiglio	Analyst	50

## Select Rows that Satisfy a Null Value Test

Use a null value test when you want to select only rows that have no value in a field.

### Syntax:

```
SELECT column_name[, column_name . . .]  
FROM table_name  
WHERE column_name IS NULL
```

### Example:

List the names and salaries of all employees who do not receive commissions.

```
SELECT ename, sal FROM employees WHERE comm IS  
NULL
```

### The Result:

Ename	Sal
Markson	3000
Jordan	4250
Walker	5500
Major	4750
Klein	3900
Barton	4200
Eiden	4700
Anton	3500
Carlisle	3375
Bonfiglio	4175

O'Neil      4850

---

Use a compound null value test when you want to select only rows that have some value in a field.

**Syntax:**

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE column_name IS NOT NULL
```

**Example:**

List the names, salaries, and commissions of all employees who receive commissions.

```
SELECT ename, sal, comm
FROM employees
WHERE comm IS NOT NULL
```

**Result:**

---

Ename	Sal	Comm
Allen	4300	2000
George	4400	3000
Turnbull	4050	5050

---

## Select Rows that Satisfy More than One Condition

Use compound search conditions, also called compound logical expressions, when you want to select rows that satisfy more than one condition. Compound search conditions are connected by the logical operators AND, OR, and NOT.

When a condition contains more than one expression, the RDBMS evaluates each expression. The results of all the conditions are then combined according to the order determined by the precedence of the connecting operators:

- Equal Precedence

`=, !=, >, > =, IN, LIKE, IS NULL, BETWEEN`

- Logical Operator Precedence

`NOT, AND, OR`

### Syntax:

```
SELECT column_name[, column_name . . .]
FROM table_name
WHERE expression
AND [NOT] (expression)
OR [NOT] (expression)
```

### Example:

List the employee name, job, hire date, and salary for each manager who earns \$4,300 or more, and for all the salespeople.

```
SELECT ename, job, hiredate, sal
FROM employees
WHERE (Sal >= 4300
      AND job = 'Manager')
      OR job = 'Salesman'
```

**Result:**

Ename	Job	Hiredate	Sal
Allen	Salesman	15-Apr-85	4300
George	Salesman	01-Jan-94	4400
Turnbull	Salesman	15-Jun-93	4050
Major	Manager	13-Mar-85	4750
Eiden	Manager	04-Aug-89	4700
O'Neil	Manager	15-May-94	4850

**Example:**

List the employee name, job, hire date, and salary for all the managers and all the salespeople who earn \$4,300 or more.

```
SELECT ename, job, hiredate, sal
FROM employees
WHERE sal >= 4300
      AND (job = 'Manager'
          OR job = 'Salesman')
```

**Result:**

Ename	Job	Hiredate	Sal
Allen	Salesman	15-Apr-85	4300
George	Salesman	01-Jan-94	4400
Major	Manager	13-Mar-85	4750
Eiden	Manager	04-Aug-89	4700
O'Neil	Manager	15-May-94	4850

# *Advanced SELECT Statements*

---

# 7

**W**ith advanced SELECT statements you can conduct queries that are more complex than the queries discussed until now.

## *Expressions*

Expressions let you modify the way fields are displayed. Using expressions you can compute values on one or more columns to create single columns in the result table, or you can modify how the comparisons are made among columns.

Sometimes expressions contain functions that have no standard syntax but nonetheless increase the power of queries. Different vendors supply different function syntax, but the way these functions are used is standard.

This chapter discusses a few of the most popular expressions and functions:

- Arithmetic expressions used for calculations
- Null functions used to manipulate Null values in reports and conditions
- Concatenation functions used to improve the way we can display columns in the result table

Each database implements a different set of functions and should be studied specifically.

## Arithmetic Expressions

Arithmetic expressions in SELECT statements cause the calculation results to be displayed as though they were columns.

### Example:

List the name, salary, commission, and total compensation for all of the salespeople whose commission is greater than 50% of their salary.

```
SELECT ename, sal, comm, sal + comm
FROM employees
WHERE job = 'Salesman'
      AND comm >.50*sal
```

**Result** - Not automatically generated by Magic:

Ename	Sal	Comm	Sal + Comm
George	4400	3000	7400
Turnbull	4050	5050	9100

When an expression or individual function refers to columns that contain a null value, the result is also null.

### Example:

List the name, salary, commission, and total compensation for employees in department 40.

```
SELECT ename, sal, comm, sal + comm
FROM employees
WHERE deptno = 40
```



## Result:

Ename	Sal	Comm	Sal + Comm
Allen	4300	2000	6300
George	4400	3000	7400
Turnbull	4050	5050	9100
O'Neil	4850		

## The NULL Functions

Sometimes you need to show or use some actual value instead of a null in reports or calculations where nulls are allowed.

For example, when calculating a statistic you may want to have a 0 instead of null, or maybe a 1. In a report you may not want to show a blank but rather asterisks or 'Not Available' to emphasize that there is no value in the column. In such cases, use an ISNULL (NVL in ORACLE 6) function, which returns the column's value if it has one, and an expression if it does not.

### NULLIF Function

```
NULLIF (expression1, expression2)
```

The resulting expression is NULL when the evaluation of (?) expression1 is equivalent to the evaluation of (?) expression2. Otherwise the resulting expression is expression1. For more details see the CASE statement in the RDBMS documentation.

### ISNULL Function

```
ISNULL (expression, value)
```

Replace NULL entries with the specified value. For example:

```
ISNULL (country, 'No Address')
```

returns the country value where it exists, and returns 'No Address' where the country value is NULL.

**Example:**

List the name, salary, commission, and total compensation for all of the employees in department 40.

```
SELECT ename, sal, comm,  
       sal + ISNULL(comm,0)  
FROM employees  
WHERE deptno = 40
```

**Result:**

Ename	Sal	Comm	Sal + ISNULL(comm,0)
Allen	4300	2000	6300
George	4400	3000	7400
Turnbull	4050	5050	9100
O'Neil	4850		4850

## ***The Concatenation Function***

Sometimes you may want a function that displays more than one column from the base table as a single column in the query result. You can use the concatenation function to combine character columns and character constants.

**Example:**

List the combined department names and locations under the same column.

```
SELECT dname || ' - ' || location
FROM departments
```

**Result:**

---

Dname    ' - '    Location
Accounting - Los Angeles
Marketing - San Francisco
Operations - Norfolk
Sales - New York
Research - Berkeley

---

## ***Multi-table Queries***

You often need to query more than one table at a time because relational architecture breaks logically connected information into physically separated tables. For example, the data of an order usually resides in two tables; one for the header and one for the order lines.

**Example:**

A company wants to list its products and how much of each product it currently has in stock. This kind of query retrieves information from two different tables: Products and Inventory. SQL lets you generate this list by using multi-table queries that join data from two or more tables.

When you build the SELECT statement you must first decide which columns you want to display. For this example you need the product names, pname, but not the product identifiers, pid.

```
SELECT pname, amount
```

You need both the Product and Inventory tables to answer the question.

```
FROM product, inventory
```

Then you need to specify how to connect the rows of the two tables, by joining the product pid and the inventory pid.

```
WHERE product.pid = inventory.pid
```

Syntax to uniquely identify each pid varies among databases, but the concept is the same.

Note: If you do not include the WHERE clause, SQL joins each row in the product table to all the rows in the inventory table.

**Stage One** - Join the product name to each inventory item.

PID*	PID*	SID*	PNAME	Amount
1	1	10	bolt	10
2	2	10	nut	20
2	2	20	nut	16
3	3	30	screw	20

\* Was not selected.

You can show any subset of columns you want, and you can choose the order among those columns through the SELECT statement.

```
SELECT pname, amount  
FROM product, inventory  
WHERE product.pid = inventory.pid
```

The first field in the SELECT list is shown first, the second second, etc.

**Result:**

PNAME	Amount
bolt	10
nut	20
nut	16
screw	20

## ***The Group Functions***

Group functions, often referred to as statistical functions, return results based upon groups of rows instead of returning one result per row.

The Group functions are AVG, COUNT, MAX, MIN, and SUM.

**Syntax:**

```
SELECT column_name,  
       function (col expression [, col expression])  
FROM table_name  
GROUP BY column_name
```

**Example:**

Display the average, highest, and total of all the annual salaries for all the salespeople.

```
SELECT AVG(sal), MAX(sal), SUM(sal)  
FROM employees  
WHERE job = 'Salesman'
```

**Result:**

AVG(Sal)	MAX(Sal)	SUM(Sal)
4250	4400	12750

COUNT(\*) - counts the number of rows.

COUNT(column\_name) - counts the number of rows where column\_name is not null.

**Example:**

List the number of rows in the Employees table and the number of employees with a non-null commission.

```
SELECT COUNT(*) , COUNT(DISTINCT Comm)
FROM employees
```

**Result:**

COUNT(*)	COUNT(Comm)
14	3

## ***The GROUP BY Clause - Grouped Queries***

```
SELECT ( ) FROM ( ) <WHERE> <GROUP BY> <HAVING>
<ORDER BY>
```

The GROUP BY clause in the SELECT statement lets you summarize query results at a subtotal level.

The GROUP BY clause divides a table into subgroups of rows. You can GROUP BY one or more columns.

- When you group by one column, all the rows with the same value for that column are grouped together. The number of groups is equal to the number of DISTINCT values found in that column.
- When you group by more than one column, a group of rows for each distinct set of columns in the GROUP BY clause is returned.

You can request group functions only on each group and each column you are grouping. Otherwise, the RDBMS may not know what to return because any other column in the group may have more than one value from different rows.

Selecting a column that is not in the GROUP BY clause without a GROUP function yields a syntax error and causes a single row to be returned for each group.

**Example:**

List each department and the number of its employees.

```
SELECT deptno, COUNT(*)  
FROM employees GROUP  
BY deptno
```

**Result:**

Deptno	COUNT(*)
10	3
20	2
40	4
50	5

**Example:**

List the number of employees in each job category in each department.

```
SELECT deptno, job, COUNT(*)  
FROM employees  
GROUP BY deptno, job
```

**Result:**

Deptno	Job	COUNT(*)
10	President	1
10	Manager	1
10	Clerk	1
20	Manager	1
20	Clerk	1
40	Manager	1
40	Salesman	3
50	Manager	1
50	Analyst	3
50	Clerk	1

## ***The HAVING Clause - Group Search Conditions***

Just as the WHERE clause can be used to select and reject the individual rows returned by a query, the HAVING clause can be used to select and reject groups of rows. The HAVING clause operates like a summary function in a subquery.

```
SELECT() FROM() <WHERE> <GROUP BY> <HAVING>  
<ORDER BY>
```

The format of the HAVING clause parallels that of the WHERE clause, consisting of the keyword HAVING followed by a search condition for groups.

**Syntax:**

```
SELECT column_name, function  
FROM table_name
```



```
GROUP BY column_name
HAVING condition
```

**Example:**

List the average annual salary for all job categories held by more than two employees.

```
SELECT job, 12*AVG(Sal)
FROM employees
GROUP BY job
HAVING COUNT(*) >2
```

**Result:**

Job	12*AVG(Sal)
Analyst	49,300
Clerk	39,500
Manager	55,500
Salesman	51,000

Grouping by job would return five groups. There are three analysts, three clerks, four managers, three salesmen and one president. The having clause limits the groups to those with more than two people, removing the president's group. It is possible to select the job for each group because it is unique for the group. There are no restrictions on group function operations, such as multiplying the average value for a group.

The HAVING clause can also be used together with a WHERE clause in the same query. The WHERE clause restricts the rows that will be included in the groups, the GROUP BY clause groups the rows, and the HAVING clause restricts the groups returned by the query.

**Syntax:**

```
SELECT column_name, function
FROM table_name
```

```
WHERE column_name = expr
GROUP BY column_name
HAVING condition
```

### Example:

List all the departments where the payroll exceeds \$10,000 excluding the clerical personnel. Order the list by the payroll amount.

```
SELECT deptno, SUM(Sal)
FROM employees
WHERE job <>'Clerk'
GROUP BY deptno
HAVING SUM(Sal)>10000
ORDER BY 2
```

### Result:

Deptno	SUM(Sal)
10	10250
50	17025
40	17600

## ***Nested SELECT Statements***

Nested SELECT statements, or subqueries, use the results of one query as the input to another query. You need to use a nested SELECT statement when the value you want to compare to a field value is known only as the answer to another query.

For example, if you want to know all the orders made by a customer named Brown, you need to write a nested query because the order table only has the customer identifier. You need to query the Customer table to find Brown's customer identifier.

While it is best to avoid nesting because of performance considerations, sometimes nesting is necessary. SQL lets you nest as many levels as you like.

**Example:** Find the oldest person in the company.

This example shows the major difference between an index sequential DBMS and an RDBMS.

- **Index sequential DBMSs** - There are two ways to conduct the search. You can scan the sequential file looking for the record with the highest age value, and then retain the name of the first person of that age for the saved record. Or you can order the file by age in descending order and take the first record and all other records with the same age.
- **RDBMS** - In an RDBMS you have no control over how the file system finds the data you need. You must treat a group of records as a group. Therefore you need nested queries so that you can use the answer to one question as input for another question.

```
SELECT ename, age
FROM employees
WHERE age = (SELECT MAX(age)
FROM employees)
```

**Result:**

Ename	Age
Major	43
O'Neil	43

Note that more than one answer is returned because two records matched the query. To limit the query to one person, you need to supply more detailed information, such as birth date.

Subqueries can be used in any condition where values can be used. If a single value is necessary, the subquery must be written to return only one value.

When the comparison operator is IN or NOT IN, the answer to the subquery may have more than one answer because these functions receive lists of values as arguments.

## Data Conversion

When using functions and comparing expressions, data types are sometimes mismatched. For example, you might request all the rows that have a '1' in a character column. The correct comparison would have to be the column compared to the string '1'. This can also happen when we compare different columns where one column might actually have numeric values and the other column might be defined as having character values.

When you want to compare columns containing different data types you need to use the SQL data conversion functions. Use the ANSI standard CAST (value AS type), where value is an expression and type is a data type, to change the type of one of the columns to match the type of the other.

### Example:

Assume that the product identifier is a character field in both the Products table and the Inventory table. Display product names for products supplied in amounts that are the product identifier multiplied by 10.

```
SELECT pname
FROM products, inventory
WHERE inventory.pid = products.pid
AND amount = 10*CAST (inventory.pid AS integer)
```

### Result:

Pname	Amount
bolt	10
nut	20

Usually the RDBMS compares columns and constants of different data types by using a default conversion function, sometimes causing unpredictable results.

**Example:**

List all employees whose hire date is before May 1, 1994.

```
SELECT *  
FROM employees  
WHERE hiredate < '1-MAY-1994'
```

The RDBMS converts '1-MAY-1994' to a date and runs the comparison.

**Result:**

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3567	Allen	Salesman	3970	15-Apr-85	4300	2000	33	40
3891	George	Salesman	3970	1-Jan-94	4400	3000	32	40
3092	Turnbull	Salesman	3970	15-Jun-93	4050	5050	40	40
3667	Markson	Clerk	3373	17-Oct-90	3000		27	50
3559	Jordan	Analyst	3408	3-May-89	4250		30	50
3123	Walker	President		13-Mar-85	5500		40	10
3472	Major	Manager	3123	2-Feb-93	4750		43	10
3373	Klein	Manager	3123	2-Dec-87	3900		41	50
3162	Barton	Clerk	3162	4-Aug-89	4200		35	20
3408	Eiden	Clerk	31620	22-Jan-92	4700		33	50
3582	Anton	Analyst	3408	15-Mar-93	3500		42	20

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3115	Carlisle			27-Jul-91	3375		22	10
3012	Bonfiglio				4175		42	50

However, if the RDBMS converts the data column to a string, you might get incorrect or unpredictable results.

**Unwanted Result:**

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3567	Allen	Salesman	3970	15-Apr-85	4300	2000	33	40
3891	George	Salesman	3970	1-Jan-94	4400	3000	32	40
3092	Turnbull	Salesman	3970	15-Jun-93	4050	5050	40	40
3667	Markson	Clerk	3373	17-Oct-90	3000		27	50
3123	Walker	President		13-Mar-85	5500		40	10
3472	Major	Manager	3123	13-Mar-85	4750		43	10
3115	Carlisle	Clerk	31620	15-Mar-93	3375		22	10
3970	O'Neil	Manager	3123	15-May-94	4850		43	40

It is not good practice to rely on defaults when comparing columns of different data types, or comparing columns to constants that are of the same data type.

## ***Programming with SQL***

A cursor must be defined to retrieve each row from a result set. The cursor's body is the `SELECT` statement that returns the result set. To retrieve rows, follow these steps:

1. Declare the cursor.
2. Open the cursor.
3. Fetch row(s) from the cursor.
4. Close the cursor.

Step 3 returns a row each time it is initiated, and the next fetch returns the next row in the result set. Note that there is no fetching backwards.

In some RDBMSs, such as Informix, there are scrollable cursors.

[This page intentionally left blank]



# Modifying Data - Insert, Delete, Update

# 8

SQL is a complete data manipulation language (DML) used not only for database queries but also for retrieving and modifying data in the database. Three basic SQL statements are used to modify the contents of a database: INSERT, DELETE, and UPDATE.

## INSERT

The INSERT statement adds new rows to a table. INSERT statements are written in two different ways to perform two different functions:

- Insert one row at a time with specific values
- Insert multiple rows selected from other tables

The INTO clause specifies the table that receives the new row. The column clause indicates the columns that receive the values. The VALUES clause specifies the data values to be included in the new row.

When the column clause is omitted, values for all the columns in the table will be specified in the VALUES clause in the order defined in the base table. In columns where null values are allowed, and in situations where you do not want to put a value in certain columns, you can either not specify the column name in the column list or specify NULL as the value.

In the following example use the INSERT statement to insert one record.

**Syntax:**

```
INSERT INTO tablename [(column1,    column2, . .
.)]
VALUES (value1, value2, . . .)
```

**Example:**

A new salesperson has just been hired and his personnel data, listed below, must be added to the Employees table.

---

Name	Jack Henry
Age	36
Employee Number	3205
Manager	O'Neil
Salary	4000
Commission	None because he is new
Dept	Sales

---

**Step 1:**

Write an INSERT statement to add Jack Henry to the Employees database.

```
INSERT INTO employees
      (Empno, Ename, Job, Mgr,
       Hiredate, Sal, Comm, Age, Deptno)
VALUES (3205, 'Henry', 'Salesman', 3970,
       '28-Jul-94', 4000, NULL, 36, 40)
```

**Result:** 1 record inserted.

**Step 2:**

Check that Mr. Henry's record has been inserted.

```
SELECT *
FROM employees
WHERE empno = 3205
```

**Result:**

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3205	Henry	Salesman	3970	28-Jul-94	4000		36	40

In the following example use the INSERT statement to insert multiple records based on another table.

**Syntax:**

```
INSERT INTO table name [(column1, column2, . . .
. . .)]
```

**Example:**

The new table called Total Inventory contains two fields: pid and total.

The INSERT statement to add the inventory for each product to an inventory into the Total Inventory table is:

```
INSERT INTO total_inventory
(pid, total)
SELECT pid, SUM(amount)
FROM inventory GROUP BY pid
```

**Result:** 4 records inserted.

# DELETE

The DELETE statement removes selected rows of data from a single table.

## Syntax:

```
DELETE FROM table_name
[WHERE condition]
```

The FROM clause specifies the table containing the rows. The WHERE clause specifies which rows of the table are to be deleted.

Warning: If you don't specify a WHERE clause in a DELETE statement, all of the rows will be deleted from the table.

## Example:

Delete from the Employees table the data of an employee who has just left the company.

### Step 1:

Verify that this is the employee to be deleted by retrieving the appropriate record.

```
SELECT * FROM employees WHERE empno = 3205
```

### Result:

---

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3205	Henry	Salesman	3970	28-Jul-94	4000		36	40

---

### Step 2:

Delete the record.

```
DELETE FROM employees
WHERE empno = 3205
```

### Result:

1 record deleted.

**Step 3:**

Verify that this employee record no longer exists.

```
SELECT *
FROM employees
WHERE empno = 3205
```

**Result:**

---

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
-------	-------	-----	-----	----------	-----	------	-----	--------

---

No data found.

## **UPDATE**

The UPDATE statement modifies the values of one or more columns in selected rows of a single table.

**Syntax:**

```
UPDATE table_name
SET column1 = value1, column2 = value2 . . .
[WHERE condition]
```

The WHERE clause selects the rows of the table to be modified. The SET clause specifies which columns are to be updated and calculates their new values.

Specify a WHERE clause that uniquely identifies a row to update a single row.

**Example:**

**Step 1:**

Upgrade the salary of employee number 3123 by 3%.

```
UPDATE employees
SET sal= 1.03 * sal
WHERE empno = 3123
```

**Result:**

1 record updated.

**Step 2:**

Confirm that the table was updated.

```
SELECT *
FROM employees
WHERE empno = 3123
```

**Result:**

---

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3123	Walker	President		13-Mar-85	5665		40	10

---

The WHERE clause should identify the rows to be updated. If you do not include the WHERE clause you might update the whole table.

**Example:**

**Step 1:**

Change the job category Salesman to Salesperson.

```
UPDATE employees
SET job= 'Salesperson'
WHERE job = 'Salesman'
```

**Result:**

3 records updated.

**Step 2:**

Verify that the table has been updated.

```
SELECT *  
FROM employees  
WHERE job LIKE 'S%'
```

**Result:**

Empno	Ename	Job	MGR	Hiredate	Sal	Comm	Age	Deptno
3567	Allen	Salesperson	3970	15-Apr-85	4300	2000	33	40
3891	George	Salesperson	3970	01-Jan-94	4400	3000	32	40
3092	Turnbull	Salesperson	3970	15-Jun-93	4050	5050	36	40

[This page intentionally left blank]



**D**ata integrity refers to the degree to which data is correct and complete in a database.

## ***Data Integrity Constraints***

To preserve the consistency and correctness of its stored data, a relational DBMS typically imposes one or more data integrity constraints. These constraints can restrict the data values that can be inserted into the database. There are several different types of data integrity constraints.

### ***Single Table Constraints***

**Required Data** - Some columns in a database must contain a valid data value in every row.

**Validity Checking** - Every column in a database has a set of values that are legal for that column. This set is called a domain.

**Entity Integrity** - The primary key of a table must contain a unique value in each row.

## ***Multiple Table Constraints***

**Referential Integrity** - A foreign key in a relational database links each row in the child table containing the foreign key to the row in the parent table containing the matching primary key value.

**Business Rules** - Updates to a database may be constrained by business rules governing the real-world transactions that are represented by the updates.

**Consistency** - Many real-world transactions cause multiple updates in a database. These updates must be consistent.

The current ANSI standard requires that the definition of these constraints be declared as part of the table creation or modification.

## ***Referential Integrity***

A column or a group of columns that is used to uniquely identify rows in a table is called the primary key of the table. A column or a group of columns in one table that references the primary key or unique key in another table is called a foreign key. Maintaining the relationship between primary keys and foreign keys is known as maintaining referential integrity.

Maintaining referential integrity is a task best relegated to the RDBMS. Each commercial vendor provides some level of referential integrity support.

As the demand for RDBMSs to maintain referential integrity evolved, so did the language for specifying it. Specifying keys and references as part of the table creation step is known as declarative referential integrity.

# Database Support for Constraints

Databases support the constraints listed below through their table declarations or through their declarative referential integrity:

**Primary Key** - A column or a group of columns that is a unique key and considered the main access route to the row.

**Unique Key** - A column or a group of columns that uniquely identifies the row. There may be one or more unique keys to a table.

**Foreign Key** - A column or a group of columns that relates a row in one table to a unique row in another table, usually by the other table's primary key.

The constraints above restrict the values of columns in the table either absolutely or in certain columns of the specific table. The current ANSI standard requires that the definition of these constraints be declared as part of the table creation or modification.

**Default Values** - A column can be given a default value that will be used when no column is specified in an INSERT statement.

## Example:

Create Table TEST1

```
(FLD1 char(1) default '1' not null unique  
FLD2 integer primary key)
```

Create Table TEST2

```
(FLD1 char (1) default '2' null check (FLD1='1')  
FLD2 integer foreign key references TEST1.FLD2)
```

# Stored Procedures & Triggers

## Stored Procedures

Database trigger technology evolved because declarative referential integrity was not available and business rules needed to be implemented. This technology, called stored procedures, is useful for maintaining all of the constraints mentioned above. Stored procedures are also useful for implementing business rules that cannot be expressed by the data structure and for reducing network traffic.

A stored procedure is a block of SQL statements that can be called by any other SQL statement, or by any other command language, including Magic. Stored procedures are usually used to enforce the business rules required by the database. In some implementations stored procedures are augmented by procedural language.

Stored procedures are sections of code stored in the database either as procedures or as functions. Different databases implement stored procedures differently. Some databases store compiled code, and others do not. The language of stored procedures is not standard, but the basic principles are the same. Any software that can access the database through SQL commands can access the stored procedures as well.

**Examples** of writing stored procedures in Oracle, MS-SQL, and Sybase.

### Oracle:

```
Create or replace procedure sp1
(p1 char in, p2 integer out)
as
begin
    update table1 set fld1 = fld1(7 where fld1 =
p1;
    select fld2 into p2 where fld1 = p1;
end sp1;
```

## Sybase and MS-SQL:

```
Create procedure sp1 @ p1 char(5)
as
begin
    insert into table1 values ('a',@p1)
    select fld2 from table2 where fld1=@p1
end
```

## *Executing Stored Procedures*

Stored procedures in some RDBMSs, such as Sybase and MS-SQL, are executed in a way that is similar to the execution of a SELECT statement.

If a SELECT statement in the stored procedure produces a multi-line result, the lines can be fetched because there is a cursor on the SELECT statement.

In Oracle, a stored procedure usually results in single lines, but multiple-line results are also possible.

### **Example:**

#### **Oracle**

```
execute sp1(p1,p2)
```

#### **MS-SQL**

```
sp1('hello')
```

#### **Result:**

```
Fld1/hello
```

## Triggers

Database triggers are database events, such as INSERT, UPDATE, or DELETE statements that trigger either SQL statements or stored procedures.

Triggers are especially useful when updating data in one place causes data to be updated in another place, as in the examples below:

- When you add a new order line to an existing order, the new line should update the running total of the order.
- When you remove items from the inventory, the stock level should be checked and a stock report issued if the amounts in stock have fallen below a certain level.

### Syntax:

```
Create trigger trig1 table1
on table1 for update
as
begin
    update table2 set    table1.fld2=
                        table1.fld2+updated.fld2
                        where
table1.fld1=updated.fld1
```

Note: Syntax for writing triggers also varies among different RDBMSs.

Certain types of transactions can be better implemented as database triggers and stored procedures than as application logic. Transactions will be defined and explained below.

**A** transaction is a sequence of one or more SQL statements, which are usually closely related but perform interdependent actions, and which form a logical unit of work. Each statement in the transaction performs some part of a task, but all of the statements are required to complete the task. The DBMS executes the sequence as one operation because the statements are grouped as a single unit. All the statements must be completed for the database to remain consistent.

When applications update multiple tables in a database, transactions ensure database integrity.

## *The Transaction Mechanism*

If the transaction does not complete successfully, a ROLLBACK statement returns the data to the state it was in prior to the beginning of the transaction. If the transaction completes successfully, a COMMIT statement permanently stores the transaction.

Due to the nature of SQL and relational database architecture, where each update can only act on one table at a time and where there are no trailers in the usual sense, two update statements must be executed simultaneously. Both update statements must either succeed and COMMIT or fail and ROLLBACK.

In a transaction a group of updates must either COMMIT or ROLLBACK together. First a transaction is declared, and then either the COMMIT or ROLLBACK SQL statement is issued for all the statements that have been issued since the transaction declaration.

## ***Two-Phase COMMIT***

Implementing transactions over distributed databases, such as updating tables that are not on the same node in a network, can create problems. A two-phase COMMIT (2PC) protocol solves this problem. Each RDBMS vendor implements the two-phase COMMIT differently in different versions.

The two-phase COMMIT is the standard way to complete transactions that update tables that are usually separated by a network and managed by separate servers in more than one database.

First a COMMIT statement is posted to all participating servers. After the posted COMMIT has been confirmed, the COMMIT is finalized on a second pass. Any network failure during this process locks the participating records. Most RDBMSs today support 2PC among databases and servers of their own type.

## ***Data Replication***

Data replication, an alternative to the two-phase COMMIT, is sometimes necessary in distributed databases. Data replication, which is less resource-intensive, is more popular when there is no need for simultaneous updates to remote tables.

Some tables have few updates, and the network architecture requires copies to be held at different nodes. Data replication can be implemented through transactions that update the copies. However, this can cause problems during network failure and may be unnecessary, especially when the data is not required to be always totally up-to-date. In such cases the copies can be updated asynchronously. Updating replicated data will not cause a ROLLBACK if there is network failure, and the copy can be updated



manually when the network is working again. Different vendors offer their own implementations of data replication.

There are different types of replication. In general, a specific type can be chosen based on several different conditions, such as:

- Both locations need to update the database
- Only one location needs to query the database
- Required frequency of location synchronization
- Procedure to be performed when there is an update conflict
- Whether or not the entire database must be replicated

## ***Locking***

Records are locked to preserve data integrity and to give each user a consistent view, while providing maximum concurrency. Locking prevents a record or group of records from being changed while a user is viewing or modifying them. Locks can either be exclusive, not allowing other users to even read the records, or shared, letting other users read the records without modifying them in any way.

Transactions and locking are tightly bound in RDBMSs. Because SQL databases run in multi-user environments, indiscriminate locking can cause an application to severely limit user access. Different levels of locking are available in all the RDBMSs.

In ISAM a record is locked for updating and then immediately released when we leave it. In an RDBMS the lock is enforced at the beginning of the transaction and released only by a COMMIT or ROLLBACK operation.

## Isolation Levels

Isolation levels determine the type of phenomena allowed to occur during the execution of concurrent transactions.

Three phenomena define SQL isolation levels for a transaction:

- *Dirty reads* allow different results to return within a single transaction when an SQL operation encounters an uncommitted or modified record created by another transaction. Dirty reads increase concurrency but reduce consistency.
- *Non-repeatable reads* allow different results to return within a single transaction when an SQL operation reads the same row in a table twice. Non-repeatable reads can occur when another transaction modifies and commits a change to the row between transaction reads. Non-repeatable reads increase consistency but reduce concurrency.
- *Phantoms* allows different results to return within a single transaction when an SQL operation retrieves a range of data values twice. Phantoms can occur if another transaction inserted a new record and committed the insertion between executions of the range retrieval.

Each isolation level differs in the phenomena it allows.

---

Isolation Level	Dirty Read Allowed	Non-repeatable Read Allowed	Phantoms Allowed
0. Dirty Reads	YES	YES	YES
1. Read Committed	NO	YES	YES
2. Repeatable Read	NO	NO	YES
3. Can be Serialized	NO	NO	NO

---

## ***Setting the Isolation Level***

Most RDBMSs use isolation level 1 by default. The user may change the isolation level in the database, the session, or in the SELECT statement level.

For example, in Sybase and MS-SQL,

```
Set isolation level = 0
```

allows dirty reads.

## ***Locking Levels***

During a normal operation, the RDBMS locks the view structures. Implicit locking is performed automatically and protects the data without any user intervention. Overriding default locking is known as explicit locking.

Implicit locking occurs automatically when SQL statements are executed. For example, the statements INSERT, UPDATE, and DELETE cause implicit locking so that data consistency and integrity are maintained during transactions.

Some RDBMSs, such as Oracle and Informix, acquire locks at a record level. Sybase and MS-SQL acquire page-level locks, which cause other records belonging to the same page to also be locked.

A lock acquired at record or page level implicitly creates a table-level lock to avoid mutually exclusive locks from other transactions. A lock can be either an exclusive or a shared lock.

## ***Escalating a Lock to a Table Lock***

In some database systems lock escalation occurs because when many locks are held at one level, the RDBMS automatically changes these locks to a different lock at a higher level (such as a table lock). The number of locks is therefore reduced, but the granularity of what is locked is increased.

## ***Enforcing Locks***

Explicit locking can be acquired at the same levels as implicit locking. When updating a record, an implicit exclusive lock is acquired at record level.

When initiating a

```
select * from table where ... for update
```

statement under Oracle, or a

```
select * from table where with holdlock
```

statement under Sybase, an implicit shared lock is acquired at record or page level, and the record cannot be updated until the lock is released.

Note that while using Informix, one cannot explicitly lock records when using the ORDER BY clause.

## ***Locking Duration***

A data lock is acquired at some time during the lifetime of a transaction and is withheld until a COMMIT/ROLLBACK command is initiated. Locks are not released during a transaction.

### **Example:**

```
begin transaction;  
update table set fld1=1 where fld2=2  
update table set fld1=3 where fld2=4  
both records are locked with an exclusive lock  
commit * both records are released
```

# *System Tables and Security*

# 11

---

**E**ach database has a data dictionary that includes all the system tables and database views. The data dictionary's objects are used as a read-only reference about the database. The RDBMS security mechanism, defined in some of the system tables, lets you define different types of users in the database, and assign and revoke user privileges according to organizational considerations.

## *System Tables*

The data dictionary is created when a database is created. The data dictionary is automatically updated by the RDBMS in response to specific actions, such as CREATE TABLE. Database operation is dependent on the data dictionary. Because data in the base tables of the data dictionary is necessary for the RDBMS to function, only the RDBMS should write or change the information in the data dictionary.

During database operation, the data dictionary is read by the RDBMS to check that the object exists and that the user has proper access to the object. No data in any data dictionary tables should be deleted or altered by any user, except for the data in the auditing tables when auditing is enabled.

For convenience, public synonyms have been created on many data dictionary views. Each RDBMS has a different data dictionary structure.

The data dictionary's views serve as a reference for all database users. Access to the data dictionary's views is via SELECT commands. The data dictionary for each database is always available when the database is open.

## ***How System Tables are Implemented***

### ***Oracle Data Dictionary***

The Oracle data dictionary consists of sets of views. In general, a set consists of three views containing similar information and distinguished from each other by their prefixes.

- USER - Describes a user's profile
- ALL - Describes a user's access rights
- DBA - Describes all the users' access rights

The set of columns is identical across all the views with the following exceptions:

- USER views usually exclude OWNER columns
- Some DBA views have additional columns containing useful information for the DBA

USER views can:

- Refer to the user's own private environment in the database
- Display only rows that are relevant for the user
- Have abbreviated PUBLIC synonyms for convenience

## ***Informix, Sybase, MS-SQL, DB2 Data Dictionaries***

The RDBMSs of Informix, Sybase, MS-SQL and DB2 also use system tables like those used by Oracle, except for the structure and the naming conventions.

In all four RDBMSs the system tables are named sys + Object\_name, where Object\_name stands for tables, views, indexes, objects, and so forth.

In Informix, Sybase, and MS-SQL, each object has an ID that is used to identify each unique row and link among tables.

Like Oracle, DB2 uses the name of the object to uniquely identify the object.

## ***How to Use System Tables***

You can use the system table to obtain all the information you need about the exact structure of the database and the database objects. You can view the:

- Content of a SELECT statement that created a view
- Data types and length of columns in a table
- Content of stored procedures and triggers

For example, if you want to view all the indexes of a specific table and you only know the table name, issue the following SELECT statements:

### **In MS-SQL and Sybase**

```
Select a.name from sysindexes a ,sysobjects b
where a.id=b.id and b.name='table_name'
```

This will join the two system tables and look for the index name from sysindexes where the id for that index is the id of a table called 'table\_name'.

### **In Informix**

```
Select a.idx_name from sysindexes a, systables b
where b.tabname = 'table_name' and
a.tabid=b.tabid
```

This will join the two system tables and look for the index name from sysindexes where the id for that index is the id of a table called 'table\_name'.

### **In Oracle**

```
Select Index_name from user_indexes  
where Table_name='Table_name'
```

This results in all the indexes of table 'table\_name'.

Note: The Magic Get File Definition operation reads the system tables and incorporates their definitions into Magic.

## ***System-Stored Procedures in MS-SQL and Sybase***

In addition to the system tables, Sybase and MS-SQL server have system-stored procedures that provide shortcuts for querying the system tables. System procedures are created in the master database during installation and are owned by the system administrator (SA). All system procedure names begin with sp.

System procedures can be run from any database. When a system-stored procedure is executed from a database other than the master database, any non-qualified references to system tables are mapped to the database where the procedures are running.

System-stored procedures are located in the master database, where system permissions are also defined. Some system-stored procedures can be run only by database owners (dbo). Other system-stored procedures can be executed by any user granted EXECUTE permission in the master database on these procedures.

For example, the sp\_help stored procedure returns all the objects in the database.

The following SQL command also returns all the objects in the database:

```
SELECT      name  
FROM        sysobjects
```



# Security

The RDBMS is a multi-user system, and therefore includes security features that control access to and use of the database. There are two kinds of database security systems: system security and data security.

## ***System Security***

System security controls access to and use of the database at the system level by checking:

- user name and password
- user authorization to connect to the database
- amount of disk space available for the user's objects
- limit of user's resources
- which system operations a user can perform

## ***Data Security***

Data security controls access to and use of the database at the object level by checking:

- which users have access to a specific object
- the specific types of actions allowed each user on the object

Every object in the database is a combination of a database.owner.table\_name. A user can access an object only if the appropriate privilege has been assigned. Each database has a list of user names. To access a database, a user must try to connect with a valid database user name. To prevent unauthorized use, each user name has an associated password.

To increase table security, you can restrict certain users to view access only without granting access rights to any database objects underlying the views. A view can provide access to selected columns of the database objects that

define the view. In addition, views can provide value-based security for the information in the objects.

## ***Using the Operating System User Name***

In Informix and Oracle, users can be checked by the operating system.

### ***Oracle OPS ACCOUNT***

If your operating system permits, you can authenticate users. Set the `OPS ACCOUNT` parameter to

```
OS_AUTHENT_PREFIX.users
```

to define a prefix that Oracle adds to the beginning of every user's operating system account name. When a user tries to connect to the database, Oracle compares the prefixed user name with the Oracle user name in the database.

#### **Example:**

```
OS_AUTHENT_PREFIX=OPS$  
Operating System account: Magic
```

Oracle looks for a database user "OPS\$MAGIC and only allows the user to connect if that database user is found. Under Oracle, all references to a user authenticated by the operating system must include the prefix, and therefore look like OPS\$MAGIC.

## ***Informix***

Before Informix Version 7, Informix always used the operating system account name as the user name. No user name was defined inside Informix. Every operating system user who could connect to Informix had a user name in Informix with the same name as in the operating system account. The user name was automatically created in Informix the first time the user connected to Informix.

In Informix 7, you can choose to define users in Informix or to use the operating system account name.

## ***Oracle/Rdb***

The user name is derived from the operating system user name and used to connect to the database.

# ***Granting and Revoking System Privileges***

You can GRANT and REVOKE permission to a specific user on a specific operation in the database.

## ***System Privilege***

```
GRANT\REVOKE {ALL\ statement.list}  
TO\FROM {PUBLIC\ name list}
```

where the statement list includes CREATE TABLE, VIEW, RULE, etc.

## ***Object Privilege***

```
GRANT\REVOKE {ALL\ permission.list}  
ON {table name [(column list)]...  
TO\FROM {PUBLIC \ name list}
```

where the permission list includes SELECT, UPDATE, INSERT, REFERENCES.

### **Example:**

- GRANT CREATE TABLE TO SCOTT.
- REVOKE SELECT ON TABLE1 FROM SCOTT.

When you are using the Windows operating system, you can use the following RDBMS programs to grant and revoke system privileges:

Oracle	User Manager of SQL & DBA, or Enterprise Manager
Sybase	SQL Server Manager (SSM)
MS-SQL	SQL Enterprise Manager
Informix	DB-ACCESS
DB2	DB2 Command Line

**T**he SQL server has an optimizer component that finds the most efficient way to access each table, resulting in quick responses to user queries. According to the kind of query, the optimizer determines whether to scan a table sequentially, use an index, sort by temporary tables, or use any other access method.

While all RDBMS optimizers are similar, every RDBMS vendor develops and improves its optimizer differently to comply with the internal structure of its SQL server.

## *How the Optimizer Works*

The optimizer has two major inputs:

- Query structure - Which tables to select from, WHERE clause, ORDER BY clause, and so forth.
- Database structure - Which columns have an index, what kind of index, and so forth.

Several rules are defined within the optimizer. A few general rules, which can be used in all optimizers, are listed below:

- The WHERE clause is more important than the ORDER BY clause.
- If a column is compared with a constant or a variable, check if there is an index on it, and if so, use it.
- A unique index is preferable to a non-unique index.
- If the columns in the ORDER BY clause match an index and do not conflict with the index that will be used for the WHERE clause, use the index to process the ORDER BY clause.
- If an index can be used for the GROUP BY clause, use it.
- If a column from one table is compared with a column from another table, check which column has an index defined on it. Use the table containing the column with an index defined on it first.

The optimizer also decides which method will give the quickest results. For each table it has the following options:

- Sequentially scan the table.
- Use an index to point to the right rows.
- Create and use a temporary index.
- Perform a sort.

## ***The Access Path***

The SQL RDBMS receives application requests in the form of SQL statements. After performing a syntax check, the RDBMS decides how to perform the required operation.

Consider the following SQL statements received by a database user:

```
SELECT A.a,A.b,A.c,B.a,B.b,B.c
FROM tableA a, tableB b
WHERE A.a = B.a and A.d =5 and B.e = 'open
requests'
ORDER BY B.b;
```

Because the statement is a valid one, and tables tableA and tableB exist and contain the mentioned columns, the RDBMS performs the statement to build the result table in response to the user query.

To perform the query the RDBMS has the following access path to the data (partial list):

1. Read all rows for table tableB. Check if the value of B.b is 'open requests'. For each of the records found, scan tableA and check if A.d is equal to 5 and A.a is equal to B.a. Sort all the records found by B.b.
2. Read all rows for table tableA. Check if the value of A.a.d is 5. For each of the records found scan tableB, and check if B.e is equal to 'Open requests' and A.a is equal to B.a. Sort all the records found by B.b.
3. Sort all records from tableA having A.d = 5 by A.a. Sort all records from tableB having B.e = 'OPEN requests' by B.a. Perform a merge for the two sorted lists and sort the result by B.b.
4. Use indexes ....

If the optimizer selects a wrong access path, the user may have to wait a long time to receive the response. Therefore it is important that the optimizer have as much information as possible about the database and its requests. System tables containing information about application tables should be updated with current statistics, and SQL statements should be checked to ensure that as much information as possible is supplied to the optimizer.

## ***Update Statistics***

An optimizer uses a set of rules to determine the access path. Some information, such as table size and index population, can be crucial in making the choice.

You can optimize tables by periodically updating index statistics on the tables. To update statistics, use the UPDATE STATISTICS statement.

### ***MS-SQL and Sybase***

```
UPDATE STATISTICS  
[database.][owner.]table_name [index_name]
```

### ***Oracle***

```
ALTER TABLE - table name  
COMPUTE\ESTIMATE STATISTICS
```

Statistics about key-value distribution in each index are kept by the SQL server. The optimizer uses these statistics when determining which index to use in query processing. Query optimization depends on the accuracy of the distribution steps.

Rerun UPDATE STATISTICS whenever the key values in the index have changed considerably.

Use UPDATE STATISTICS when:

- a large amount of data in an indexed column has been added, modified, or removed
- the table has been truncated using the TRUNCATE TABLE statement and then repopulated

When an index is created or recreated on a table that already contains data, UPDATE STATISTICS runs automatically.



## ***Hinting the Optimizer***

Sometimes the optimizer performs a wrong selection. Most RDBMSs have an option that lets you force the optimizer to use a specific access path. This option should be used only by an expert who understands why the optimizer performed incorrectly. Inappropriate use of this option may cause a change in response time.

## ***Track the Optimizer Decision and Access Path***

### ***MS-SQL and Sybase***

#### ***Set Showplan on Command to See***

Use the plan for the query. You can use this option to see whether an index is being used to retrieve query results. To understand the Showplan output, refer to *The Administrator Companion*, Chapter 23, Understanding Showplan Output.

You can execute this option by writing it as an SQL command or by clicking the Dbplay Showplan icon found at the bottom left of ISQL/W.

It is often preferable to focus exclusively on the statistics to output and experiment with different query and index types than it is to use Set Showplan.

When the database is changed by events such as adding indexes, stored procedures must be optimized again by recompiling them. An automatic optimization will be performed only after the SQL server has been restarted and the stored procedure is already running, or when the underlying table used by the procedure changes. To recompile stored procedures that are related to a table, use the following command:

```
sp_recompile  
table_name
```

# Oracle

## **Oracle SQL Trace Facility**

The SQL Trace facility generates statistics for each SQL statement.

The SQL Trace facility can be defined at instance level or at session level. When the SQL Trace facility is enabled, performance statistics for all SQL statements executed in an instance or in a user session are placed into a trace file.

A program called TKPROF formats the trace file contents and writes the output into a readable output file. TKPROF can also determine the SQL statements' execution plans and create an SQL script to store the statistics in the database.

## **Enabling the SQL Trace Facility**

Enabling tracing for an instance is done by setting the value of SQL\_TRACE to TRUE. This value causes statistics to be collected for all sessions.

Enabling tracing for a session is done by the SQL command:

```
ALTER SESSION
SET SQL_TRACE = TRUE
```

If the trace has been enabled for an instance, it may be disabled for an individual session by using the following SQL command:

```
ALTER SESSION
SET SQL_TRACE = FALSE
```

When the trace is enabled for an instance, Oracle creates a separate trace file for each session.

## **Running TKPROF**

TKPROF produces a formatted output file from a trace file that is produced by the SQL Trace facility. You can use TKPROF to generate execution plans as well.

```
TKPROF input-trace-file output-file EXPLAIN=
username/password
```

Use EXPLAIN username/password to determine the execution plan for each SQL statement in the trace file and to write these execution plans to the output file.

For more information about TKPROF and its parameters, refer to *Oracle 7, Server Application Developer's Guide*, Appendix B, Performance Diagnostic Tools.

### **EXPLAIN PLAN Command**

To display the execution plan chosen by the Oracle optimizer for SELECT and DML statements, use the EXPLAIN PLAN command. The execution plan of a statement is the operation sequence performed by ORACLE to execute the statement.

There must be a table to hold the EXPLAIN PLAN output before the EXPLAIN PLAN command is executed. You can create the table manually by using the CREATE TABLE command or by running the UTLXPLAN.SQL script. The table can have any name, but the column names and data types must be the same as in PLAN.TABLE.

```
EXPLAIN PLAN
[SET STATEMENT_ID='text' ]
[INTO table]
FOR statement;
```

To select information from the explain plan table, you can use the following SELECT statement:

```
SELECT LPAD (' ',2*(LEVEL-1))
      operation _ operations,
      options, object_name, position
FROM plan_table
START WITH id=0
AND statement_id = 'statement id from EXPLAIN
PLAN command'
CONNECT BY PRIOR id=parent.id
AND statement.id = 'statement id from EXPLAIN
PLAN
command'
```

For more information about the EXPLAIN PLAN command, refer to the *Oracle 7 Server SQL Language Reference Manual*.

## **Informix**

### ***Enabling Set Explain On***

Enabling the Set Explain On option lets the optimizer access path be traced for a session performed by an SQL command.

### ***Set Explain On***

A file called sqexplain.out is created in the default directory, and all commands and access paths are written to this file. The data in the sqexplain.out file is easy to understand and use.

## ***Oracle/Rdb***

Define

```
RDBMS$DEBUG_FLAG sto
(s-strategy,
t-attach, c-cost)
```

Define

```
RDBMS$DEBUG -FLAGS_OUTPUT filename
```

where the flags can be various flags. This definition ensures that all the information will be written to the specified file.

Refer to the relevant RDBMS documentation for more information.

## ***Examples***

A representative example from Microsoft SQL server follows:

Assuming that the SQL commands below have been entered and the EMP table has been created, a few SELECT statements are analyzed below.

### ***SQL Command to Create the Data***

```
create table emp ( empno integer , ename
char(20), deptno smallint)
create unique index empno_ind on emp (empno)
create index empname_ind on emp (ename)
insert into emp values (1,'tomer',2)
insert into emp values (2,'dalit',2)
insert into emp values (3,'osnat',4)
insert into emp values (4,'michel',1)
insert into emp values (5,'john',3)
insert into emp values (6,'benjamin',1)
```

## Simple *SELECT* Statement

```
SELECT * from emp WHERE empno>1
```

**Result:**

### Step 1

The type of query is *SELECT*.

```
FROM TABLE  
emp  
Nested iteration  
Index : empno_ind
```

empno	ename	deptno
2	Dalit	2
3	Osnat	4
4	Michael	1
5	John	3
6	Benjamin	1

The index empno\_ind was used to retrieve the records.

## ***SELECT Statement with WHERE and ORDER BY Clauses***

```
SELECT *  
FROM emp  
WHERE empno>1  
ORDER BY ename asc
```

### **Result:**

#### **Step 1**

The type of query is INSERT.  
The update mode is direct.  
Worktable created for ORDER BY

```
FROM TABLE  
emp  
Nested iteration  
Index : empno_ind  
TO TABLE  
Worktable 1
```

#### **Step 2**

The type of query is SELECT. This step involves sorting.

```
FROM TABLE  
Worktable 1  
Using GETSORTED Table Scan
```

empno	ename	deptno
6	Benjamin	1
2	Dalit	2
5	John	3
4	Michael	1
3	Osnat	4

Because the WHERE clause takes priority over the ORDER BY clause, the SQL server first used the index empno\_ind to specify the WHERE clause, and then used a temporary work table to perform the sort.

All the employee numbers are greater than 1. Therefore the use of EMPNAME\_IND results in a better response time, so we can give the optimizer a hint to use that index.

```
SELECT * from emp(index=empname_ind) where empno1  
order by ename
```

### **Result:**

#### **Step 1**

The type of query is SELECT.

```
FROM TABLE  
emp  
Nested iteration  
Index : empname_ind
```

empno	ename	deptno
6	Benjamin	1
2	Dalit	2
5	John	3
4	Michael	1
3	Osnat	4



---

**M**agic and SQL databases complement each other and are strategic partners in the ongoing effort to keep software costs under control. Applications developed in Magic are used with a growing list of SQL-compliant RDBMSs, including Sybase, MS-SQL, DB2, Oracle, and Informix.

Magic can be used with an underlying SQL database to rapidly create robust, real-world business applications. In the remaining sections of this guide, you will see how SQL's weaknesses are precisely Magic's strengths, and how SQL's strengths can be selectively and conveniently incorporated into the Magic environment.

## ***Why Use SQL with Magic***

Magic lightens the burden of programming with language-based development tools. SQL acts as a common standard for accessing databases and thereby reduces the indirect costs of learning, training, and maintaining applications that access multiple databases.

Magic supports many types of SQL databases. By using SQL with Magic, you can develop, support, and maintain portable and interoperable SQL database applications. In the past you would have used embedded SQL or dynamic SQL in conjunction with 3GL or 4GL tools when application functionality could not be achieved using SQL, or when the functionality was too

cumbersome to use SQL. Now Magic provides a more productive and effective tool.

Magic and SQL are natural partners for developing strategic and cost-effective MIS applications. Magic's support of SQL-compliant databases is both thorough and flexible, enabling the best features of SQL to be easily incorporated into Magic applications while compensating for specific SQL database deviations from the ANSI standard.

The Magic runtime engine is SQL-aware and has an internal mechanism that was constructed using SQL concepts and terminology. Each Magic SQL gateway is also specifically designed to accommodate the differences among RDBMS implementations. The developer does not need to understand those differences because Magic makes the necessary adjustments. Elements of this internal mechanism are explained in the next section.

## ***Magic SQL Support***

Magic provides access to various SQL-based Relational Database Management Systems (RDBMS) using the MagicGate Database Gateway products. Magic gateways are available for major RDBMS systems such as Oracle, MS-SQL, Sybase SQL Server, Informix, Oracle/Rdb, DB/2, and ODBC.

Magic supports SQL at both the implicit and explicit level.

### ***Built-in Support - The Implicit Level***

With Magic's built-in SQL support, a developer does not have to write any SQL statements. Magic manages all SQL activity transparently. The developer is separated from the underlying DBMS and can concentrate on building the application.

Magic interfaces with SQL databases using MagicGate Database Gateways. Each database gateway is built to accommodate the unique features and syntax of the SQL DBMS it interfaces with. Magic can load multiple database gateways simultaneously and perform cross-DBMS joins in a single program.

## ***Embedded SQL - The Explicit Level***

Magic defines embedded SQL literally as SQL statements embedded into a Magic task. With Magic's embedded SQL support, the developer can provide the SQL statements explicitly and have them transferred to the underlying DBMS for processing. Magic then manipulates the results of these SQL statements.

## ***Magic & SQL Terminology***

The way certain terms are used in Magic may differ from standard SQL terminology. Refer to the following table for a translation of terms:

Magic Term	SQL Term
File	Table, Relation
Record	Row
Field	Column
Link	Join, Nested SELECT
Dataview	Table, View, or Join

## ***Magic Gateways Naming Conventions***

To structure the Magic Gateways version and to be able to work with future versions, the naming conventions of the gateways have changed.

- The first string will contain the RDBMS name and version.
- The next string will be the word version.
- The third string will be the Magic version number followed by - character.
- The last string will be the source id which will be in the format of X.Y, with X representing the major source version, such as 6,7, or 8, and Y representing the minor source version, or its running number, which is changed every time the source is changed.

For example: **Oracle 7.3 version 7.10-7.15**

Oracle 7.3 is the RDBMS name and version, followed by the word version, followed by the Magic version number 7.10, and finally the source id 7.15.

In the case of **Informix 7 version 6.02-7.4**

Informix 7 is the RDBMS name and version, followed by the word version, followed by the Magic version number 6.02, and the source id 7.4.

**Note:** This new naming convention replaces the old naming conventions.

## ***Magic's API Implementation and Versions***

Magic is a dynamic tool with no compilation or link stages. Magic programs can be executed as soon as the operation is inserted into the proper Magic table. The dataviews defined in Magic are also dynamic, and can change from one program to another and from one application to another. Magic does not limit the variety of dataviews.

To provide this versatility with a single rule-based engine, Magic uses dynamic SQL to construct the SQL dataviews that Magic programs require. Each Magic task that uses SQL tables issues the proper SQL statement to

specify the dataview needed. The SQL statement is then prepared by the MagicGate Database Gateway. During this preparation stage the DBMS optimizer analyzes the statement and prepares the statement for processing.

Unlike 3GLs and 4GLs that use embedded SQL to eliminate runtime SQL statement parsing and optimization by pre-compilation, Magic does not use compilation. Therefore, the preparation stage may produce additional overhead by executing a task that is repeatedly called. However, if the task is declared resident, the SQL statement is prepared just once and is not released. Repeated calls to the task will use the existing statement instead of preparing a new one. In this case, the performance of Magic's dynamic SQL implementation is the same as the performance of an embedded SQL program.

### **Oracle**

The Oracle Gateway is written with the Oracle Call Interface (OCI) procedures, to achieve maximum capabilities and performance. It is designed to work with Oracle version 7 and above.

**Note:** the gateway requires Oracle Pro\*C or SQL\$NET to be installed.

### **Sybase**

The Sybase gateway is written with the Sybase Library (Ctlib), and uses Ctlib commands. The gateway is designed to work with Sybase version 10.x and 11.

### **MS-SQL**

The MS-SQL gateway is written with the SQL server library (Dbllib). For maximum performance it uses client cursors and commands. It is designed to work with the Microsoft SQL server Versions 6 and 6.5, which includes the service packs.

### **ODBC**

The ODBC gateway is written using the ODBC 2.00 API. It is designed to work with ODBC driver version 2 and above.

### **Informix**

The Informix gateway is written using Esq/C API. It is designed to work with Informix version 7.x and above.

## **DB2**

The DB2 gateway is written using the DB2 Call Level Interface (CLI) procedures. It is designed to work with the DB2 version and above.

## ***Magic Repository and Capabilities***

Previous chapters of this book introduced the elements of SQL programming. This chapter will introduce some comparable programming elements in Magic.

### ***Magic Repository and SQL Data Dictionaries***

Magic's data definition capability is implemented by several repositories that work together and enable the programmer to define and describe data elements, their attributes, and their common usage.

Unlike ISAM file managers, SQL databases have their own data repositories and data definition language. SQL data dictionaries contain the names of all the tables, columns, column types, user rights, and all information about the database's logical architecture.

The power underlying Magic's data definition table structure is extensive and includes:

- **Magic's Type Repository**, which lets the developer globally define, name, and encapsulate application-specific data types. These data types, used in Magic's Column repository, are constructed from a basic set of data attributes and can include range checks, that are equivalent to SQL's domain integrity constraints, popup help screens, and automatic prompt lines.
- **Magic's Table Repository**, which lets the developer specify each individual database type and the network server address (equivalent to the SQL path) for each file (equivalent to the SQL table). This powerful and

flexible definition mechanism makes Magic applications inherently heterogeneous at the database definition level. Changing a single repository entry can completely reconfigure an application.

- **Magic's Column and Index Repositories**, which provide mechanisms for mapping SQL-specific properties for each column and index between Magic's repositories and the SQL DBMS Data Repository. These properties include RDBMS internal storage type, SQL Column Name, and SQL Index Name.

## ***Magic's Data Manipulation Capability***

Magic's data manipulation capability manifests itself in several ways, both directly and indirectly.

Magic's record-retrieval capability, the equivalent of SQL's SELECT statement, is specified directly and explicitly in table entries. Magic's Application Engine translates the table entries into the appropriate SQL DML statements.

Magic's record-manipulation capability, the equivalent of SQL's Insert, Update, and Delete statements, is specified implicitly through the task mode, which can be Create, Modify, or Delete. Magic's Application Engine detects the modes and translates them into the appropriate SQL DML statements.

This powerful combination of Application Engine and Task Modes represents one of Magic's most innovative capabilities and is a major factor in delivering massive productivity gains.

When accessing SQL databases through conventional procedural languages, sequential record processing is programmed using cursors combined with explicit calls to database command routines. Cursors are like pointers to an SQL statement. Magic programmers do not need to work in code of any sort, especially not the cumbersome cursor programming code usually required for writing RDBMS applications. Magic issues all the necessary calls to cursor routines internally.

Because Magic processes data at the record level, the Magic engine is responsible for all low-level record operations, such as opening and

maintaining cursors on the data, fetching the records from the DBMS, and inserting, updating, and deleting rows from the database tables. The developer never deals with cursor concepts and operations because Magic handles these automatically.

Using an SQL DBMS with standard Magic programming methods is similar to using Magic with an ISAM-type database. The Magic methodology remains the same and the developer's activities are similar because of the intervening layer of the MagicGate database gateways.

## ***Magic & SQL Features and Benefits***

### ***Cursor Routines***

Magic programmers do not have to deal with any of the cumbersome cursor programming usually required when writing RDBMS applications in 3GLs and 4GLs. Magic issues all the necessary calls to cursor routines internally.

### ***MagicGate Database Gateways***

Magic facilitates development in native and Client/Server configurations with a wide range of database access features. MagicGate gateways process each database request in the most appropriate way for the specific RDBMS. Many commercially available RDBMSs extend ANSI standards, but MagicGate gateways address these variations with both implicit and explicit support.

### ***Transaction Processing***

SQL databases support transaction processing. Magic brings a simple, logical approach to transaction processing by shifting the entire programming burden from the programmer to Magic. Instead of writing SQL transaction processing statements, the programmer sets flags that tell Magic at which logical level to issue the appropriate transaction processing commands. Magic issues these commands automatically and transparently. Magic also issues the actual SET TRANSACTION commands, and COMMIT and ROLLBACK operations where necessary.



## ***Managing Files and Tables***

Large organizations can use Magic's open architecture to combine data and programs from installed databases and platforms into a single, tightly integrated application development or deployment system.

When using index sequential files, Magic manages the repository describing those files. In SQL, the RDBMS has its internal accessible data dictionary. Two descriptions of the database are therefore available when writing applications that use SQL tables.

[This page left intentionally blank]

**T**here are many definitions of the Client/Server concept in the computer world today, and a wide range of solutions are called Client/Server solutions.

In general, a Client/Server configuration divides data processing tasks between the user's workstation, called the client, and the host computer, called the server. The balance between the client and the server may vary from one extreme to another.

The tasks can be divided into three major categories:

- Application logic
- Data management
- Presentation - Interface

For our purposes, the client usually handles the user interface and the application logic, or part of it, and the server handles most of the data management.

In most RDBMSs, stored procedures and other SQL server features allow more of the application logic to be moved to the server, thereby reducing the network traffic between the client and the server.

A communication layer connects the client and the server. Several standard communication protocols are used to make this connection, including:

- TCP/IP - The most well known communication protocol for UNIX, VMS, and NT server environments

- DecNet - A protocol used mainly for VAX/VMS and Alpha/OpenVMS platforms
- SPX/IPX - a protocol used in Novell networks
- Named pipes

To work with Magic and SQL databases you must understand both the Magic Client/Server architecture and the RDBMS Client/Server architecture.

## ***Magic Client/Server Architecture***

In the Magic Client/Server architecture a Magic server is installed on the server machine, and a Magic dispatcher waits for requests from clients on a known port.

When a client asks for a connection, the dispatcher identifies the client and initiates a Magic server process that represents the client on the server. From this point forward, the client only communicates with the Magic server process.

The dispatcher then continues to wait for more requests.

On the client side you need:

- TCP/IP or DecNet protocol installed and working on the client
- A communication gateway with Magic installed
- A definition SQL gateway with Magic installed

## ***Communication Gateway***

The communication gateway is supplied with the client kit that handles communication between the Magic client and the Magic server.

Magic supports the TCP/IP and DECNET communication protocols.

Under Windows the Magic communication gateway is called MGWSOCK.DLL because it uses Windows sockets.

## ***Definition SQL Gateway***

Magic uses the definition SQL gateway, which controls the file checks and some other parameters, to define the RDBMS in Magic.

When using the definition gateway, you must define your database to be on a Magic server in the Databases section of one of the following files:

- MAGIC.INI file
- mgene under UNIX
- MGSRVR.INI under VMS

This definition is necessary because it is the Magic server that accesses the database, and the real database does not reside on the client.

The definition gateway prefix is md, and the name of the RDBMS follows. For example : **mdoracle,mdora32.dll**.

## ***Full SQL Gateway***

On the Magic server side, the MagicGate Database Gateway for SQL Server must be installed and defined in Magic.

The gateway prefix is mg, and the name of the RDBMS follows. For example, **mgoracle,mgor73nt.dll**.

## ***RDBMS (Open) Client/Server Architecture***

All RDBMS vendors let you connect from a client, which can be any machine such as a PC, UNIX, VAX, etc., to a remote database that resides on another computer. Software that is installed both on the client and on the server, where the database resides, is used to make this connection. A communication protocol is also defined for both the client and the server, as in the Magic Client/Server architecture.

Each RDBMS vendor supplies a different tool:

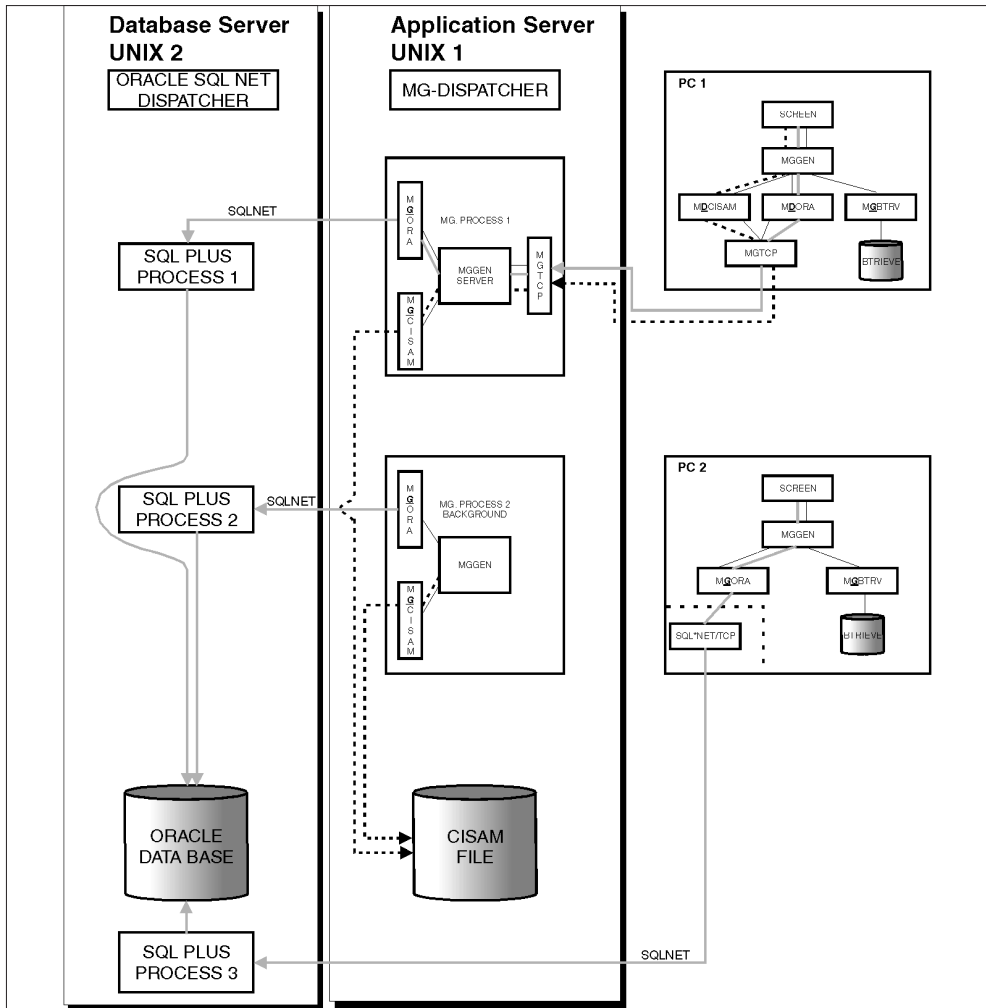
- Oracle - SQL\$NET (V1 and V2)
- Informix -INET
- MS-SQL and Sybase -NETlib
- DB2 - a DB2 Sqliib client

Each tool supports various communication protocols.

On the client side you need:

- A communication protocol supported by SQLNET, INET, and others installed and working on the client
- One of the above tools, depending on the RDBMS used
- The MagicGate Database Gateway for SQL Server

# A Heterogeneous Client/Server Configuration



The SQL Plus process is different when a multi-thread server is used.

## ***When to Use What***

The main differences between the Magic Client/Server architecture and the Open Client/Server architecture are:

- Every client in the Open Client/Server architecture should have both the RDBMS client installed and the MagicGate Database Gateway for SQL Server installed. Under the Magic Client/Server architecture, only the Definition gateway and the Communication gateway are needed.
- Under the Magic Client/Server architecture, you can do much more than just access the data. Because you have a Magic process on the server, you can print, access ISAM files, and perform many more operations on the server.
- Working in a Magic Client/Server architecture allows you to partition your application and to run background tasks on the server.
- In an Open Client/Server environment, Magic is not needed on the server because the RDBMS Client/Server handles the communication. However, Magic must be configured and installed on the client.

Use the Open Client/Server architecture when a Magic server does not exist.

When the database is installed on your PC, you are using an Open Client/Server architecture where the server is on the same machine as the client.



# *Defining the RDBMS Environment*

# 15

---

**T**o define the RDBMS environment, you must understand your RDBMS client and server configuration, and know how to check the database connections.

## *Setting the RDBMS Environment Variables*

### *Sybase 10/11*

#### ***SYBASE - Sybase home directory***

In addition to this environment variable, when Sybase is installed on Windows, you need to define the following variables:

INCLUDE - Sybase Include files directory  
LIB - Sybase library directory  
SQL10 - Sybase home directory

## **Oracle**

*ORACLE\_HOME* - Oracle home directory

*ORACLE\_SID* - Oracle instance's name.

*This parameter does not exist when working with Personal Oracle.*

In addition to the environment variables, when using SQL\*NET2, the following variable needs to be defined on the server:

`TNS_ADMIN` - SQL\*NET2 administration directory

When the Oracle Server is installed on NT, there is no need to define any environment variable.

## **Informix**

*INFORMIXDIR* - Informix home directory.

*SQLEXEC* - Informix executable or relay module

This variable is not always required.

In addition to this variable, when using Informix 7, the following variables need to be defined:

`INFORMIXSERVER` - Informix server name

`DBPATH` - The data location - for SE only

## **MS-SQL**

No variables need to be defined when using MS-SQL.

## **DB2**

When you install the DB2 client, all relevant environment variables will be defined.

## ***Connecting to the Database***

As a Magic user you can connect to the database only if the connection to the database is available from the RDBMS tools. After connecting to the database, check the user's permissions. If the permissions are wrong, ask the DBA to change the permissions accordingly.

### ***Sybase***

Add Sybase working directories to your path:

- All platforms - SYBASE\bin
- Windows platforms only - SYBASE\dll

### ***UNIX***

You can connect to Sybase using:

```
isql -Username -Ppassword -Ssybase-server-name
```

### ***Windows 3.11***

Connect to Sybase by double clicking the Isql/w icon. Fill in the Sybase username, password, and server that you are going to check.

### ***Windows 95, Windows NT***

Connect to Sybase by double clicking the Wisql32 icon. Fill in the username, password and server of Sybase that you are going to check.

Check the connection to the Sybase database by executing the following commands from isql / isql / wisql32:

```
use database-name  
go
```

## **Oracle**

Add the Oracle working directory ORACLE\_HOME\bin to your path.

### **Native Oracle**

Connect to Oracle using Sql\*plus:

```
sqlplus username/password
```

If you are using Windows, double click the Sql\*plus icon and fill in the username and password of the Oracle user.

SQL\*NET is an Oracle open client tool.

### **Using SQL\*NET1**

Connect to Oracle with SQL\*NET1 using the following command:

```
sqlplus username/password@SQL*NET1-connect-string
```

If you are using Windows, double click the Sql\*plus icon and fill in the user name and password of the Oracle user, and the SQL\*NET1 connect string.

If you are using Oracle server for NT as your Oracle client and server, you have to fill in the Host String field.

The SQL\*NET1 connect string is:

```
network-protocol:host-name:SID
```

where network-protocol is a letter that identifies the network protocol, such as:

2 - Personal Oracle, d - DECnet, t - TCP/IP, x - SPX/IPX, p-pipe name,

and host-name is the Oracle server's host name,

and SID is the Oracle instance's SID.

## ***Using SQL\*NET2***

Connect to Oracle with SQL\*NET2 using the following command:

```
sqlplus username/password@SQL*NET2-service-name
```

If you are using Windows, double click the Sql\*plus icon and fill in the Oracle user name and password, and the SQL\*NET2 connect string.

If you are using Oracle server for NT as the Oracle client and server, you must fill in the Host String field.

The SQL\*NET2 service name is defined in ORACLE\_HOME/network/admin/tnsnames.ora file.

If the Oracle server is under Windows, double click the Database Manager icon and click on the Alias ... push-button.

## ***Informix***

Add the Informix working directory (INFORMIXDIR\bin) to the path.

### ***Native Informix***

Connect to Informix using dbaccess.

- Informix 7 - In the first menu, choose Connection to connect to the Informix server, and then choose Connect. You will see a list of the Informix servers. Choose the server you want to use. After choosing the server, you will see a list of the Informix server's databases. Choose the database you want to work with.

### ***Using Inet***

Inet is an Informix open client tool.

Double click the SetNet icon. Fill in the following Informix configuration information:

- Hostname - Informix server host name  
Enter LOCAL if you use local Informix on your PC

- Username - User name in the host machine authorized to work with Informix
- Servicename - Informix service name as defined in the Services file
- Protocol name - The network protocol (TCP/IP, Named Pipe etc.)
- Password - Informix host's user password

When the definition is finished, double click the Login icon to check the connection.

## ***MS-SQL***

Add the MS-SQL working directories (SQL60\binn and SQL60\dll) to the path.

Connect to MS-SQL by double clicking the ISQL/w icon. Fill in the MS-SQL user name, password, and server you are going to check.

Check the connection to the MS-SQL database by executing the following commands from ISQL/w:

```
use database-name  
go
```

## ***Giving Database Permission***

The user can connect to the database only with permission to do so. To give the user connect permission, use the following SQL command:

```
GRANT CONNECT TO user-name
```

To change the data dictionary's data, the user needs resource permission. To give the user resource permission, use the following SQL command:

```
GRANT RESOURCE TO user-name
```

## ***Using Windows Icons***

### ***Sybase***

Double click the SQL Server Manager icon in the Sybase for Windows NT group. Enter your login name and password. Double click the Database Name icon, and then double click the User Name icon to view the user folders. Move among the folders to grant the relevant permissions.

### ***Oracle***

Double click the User Manager icon in the Database Administration Tools group. Click Create to create a new user. Click Privileges ... to grant or revoke user privileges and roles.

### ***MS-SQL***

Double click the SQL Enterprise Manager icon. Click the Manage Logins icon to define the user and the database that the user is allowed to connect to. Click the Manage Database icon to give the user authorization for DML and DDL commands.

[This page left intentionally blank]



# *Configure and Define the Magic Environment*

# 16

**T**o define the Magic environment, you must know how to define the flags and settings in the MAGIC.INI file when working with SQL databases.

## *The MagicGate Database Gateway for SQL*

### *Configuring the MagicGate Database Gateway for SQL*

The MagicGate Database Gateway for SQL must be defined and installed under both the Magic Client/Server architecture and the RDBMS Client/Server architecture. It is necessary to define for Magic that a specific gateway must be loaded, by pointing to a variable that consists of a DB number. The DB number points to a specific executable that is the relevant gateway.

- In Unix and VMS operating systems, an environment variable points to the executable which should be used for a specific Gateway.  
For example, in Unix OS:  
`MAGIC_DB_14_DRIVER=$MAGIC_HOME/bin/mgoracle8`  
where the number 14 refers to the DB number +1.
- In Windows 95 and Windows NT, the MGDBnn is set to point to the relevant DLL in [MAGIC\_GATEWAYS] section of the MAGIC.INI file.  
For example, the Oracle id in the DBMS section is 13:

MGDB13=mgor7295.dll

If you used the Magic client server, MGDB13 should refer to mdora32.dll. An example of the gateway section in the MAGIC.INI file under Windows is below:

```
[MAGIC_GATEWAYS]
;MGCOMM01=mgwsock.dll
MGDB00=\MGBTRV.DLL
;MGDB02=mdrmsw.dll
;MGDB03=mdcisam.dll
;MGDB04=mgdb3_32.dll
; MGDB05=mdrdbw.dll
;MGDB06=mgdb4_32.dll
;MGDB07=mgfox32.dll
;MGDB08=mgclp32.dll
MGDB09=mgisy32.dll
MGDB13=mgor7295.dll
MGDB14=mginf5nt.dll
;MGDB18=mddb2.dll
;MGDB19=mgodbc32.dll
MGDB20=mgms632.dll
MGDB21=mgmemory.dll
```

## ***The Magic Gateways Naming Convention***

The actual gateway image will vary from one operating system to another. Therefore, the name of the image is structured to be able to distinguish among them.

- In the Windows operating systems, the structure is as follows:
  - MXrdbmOS.DLL
    - where MX stands for MD for the definition gateway or MG for the full gateway;
    - rdbm stands for the specific RDBMS, such as INF7 or ODBC;

- OS stands for the operating system, where the values may be 32, NT, or 95  
32 means that the gateway may run on either WinNT or Win95  
NT means that the gateway can run only on WinNT  
95 means that the gateway can run only on Win95.

Note: To support different Oracle versions, the gateway must be linked to the correct Oracle libraries. Therefore, when using the Oracle gateway, the RDBMS should be specified as *ORnn* where *nn* stands for the Oracle required support files on the client, specifically 7.2 or 7.3. For example, MGORA73NT.dll is an Oracle 73 Gateway that uses WinNT.

- In the Unix operating system, the structure is as follows: mxrdbms, where:
  - mx stands md for the definition gateway, or mg for the full gateway
  - rdbm stands for the specific RDBMS, such as Informix or Oracle 73.

## ***The Magic Gateway Version Convention***

To structure the Magic Gateways version to be able to work with future versions, the naming convention of the gateways is structured as follows:

- the first string contains the RDBMS name and version number,
- the second string contains the word version,
- the third string is the Magic version number followed by the \_ character,
- the last string is the source id that will appear in the format of x.y, with x representing the major source version, such as 6, 7, or 8, and y representing the minor source version, or its running number, which is changed every time the source is changed.

For example:

1. Oracle 7.3 version 8.00-8.15  
Oracle 7.3 is the RDBMS name and version number, followed by the word version. The gateway is for Magic 8.00, and the source id is 8.15.
2. Informix 7 version 6.02-7.4  
Informix 7 is the RDBMS name and version number, followed by the word version. The gateway is for Magic 6.02, and the source id is 7.4.

## ***The [MAGIC\_ENV] Flags Section***

### ***ISAM Transactions***

Magic always uses the services of the underlying database for transaction processing. However, when ISAM files are utilized, transaction use is optional. Whenever a transaction is defined in the program, Magic will check the database of each table participating in the program. If the table originates from an ISAM database, then based on the flag, Magic will decide whether or not to apply transaction processing. The recommended value setting is YES.

When working with SQL databases the Data Recovery flag should be set to YES to enable transactions and to allow locks to be held throughout the duration of the transaction.

Because the [MAGIC\_ENV] flag is global, it will also affect transactions on Btrieve files.

The recommended setting value is YES.

## ***Display Full Messages***

The underlying RDBMS may return errors, such as constraints violations, during the application execution. Magic keeps a buffer containing the last error received from the RDBMS.

Setting the Display Full Messages flag to YES causes Magic to display the error message on the screen in Runtime, and to clear the buffer.

If you do not want the user to see these error messages, change the Display Full Messages flag to NO. Then you can use the Magic DBERR (dbname) function to put the error message into a variable as a string, and to manipulate the error message in the program.

Note that setting the Display Full Messages flag to YES causes the DBERR function to return an empty string because the error messages buffer is cleared when messages are displayed.

The recommended setting value depends on the application.

## ***Multi-user***

The Multi-user setting enables the SQL gateway to perform a logical or physical lock in the RDBMS.

In Oracle, Informix, and DB2, a FOR UPDATE clause is added to the SELECT statement that is issued when a record lock is requested.

It is advisable to set the multi-user setting to YES, especially when not using Magic locks or when applications other than Magic are accessing the database.

The recommended setting value is YES.

## **The [MAGIC\_DBMS] DBMS Section**

### ***Nulls***

When creating new rows in a Magic table, if Nulls is set to YES the default allows null in a specific column. The gateway then creates nullable columns when issuing the CREATE TABLE statement.

It is advisable to set the Nulls setting to NO and to change the setting in the Column Properties dialog only when a nullable column is required.

The recommended setting value is NO.

### ***Default Float Picture***

The Default Float Picture flag is only relevant to the Get Definition operation.

Some numeric attributes, such as float, have no length. When performing the Get Definition operation, these columns will automatically be defined according to their defaults. These defaults may be overwritten and changed in the field to the required size.

The recommended setting value is N/A.

## ***DBMS Properties***

---

Name	Type	Default	Applicable Gateways
Show Plan	Yes/No	No	Sybase, MS-SQL, DB2
Log Level	None, Developer, Support, Customer	None	All
Log Name	A255	N/A	All
Log Synch	Yes/No	No	All
Max. Connections	Number	3	MS-SQL, Sybase
Isolation Level	Number	MS-SQL=0 Sybase=1 DB2=1 Informix=1	MS-SQL, Sybase, Informix, DB2
Check Existence	Yes/No	No	All

---

## ***DBMS Parameters***

Different parameters can be specified as DBMS parameters in the *Settings/DBMS* table shown in Figure 16-1. These DBMS parameters are also applicable to all the databases defined in Magic for the specific DBMS.

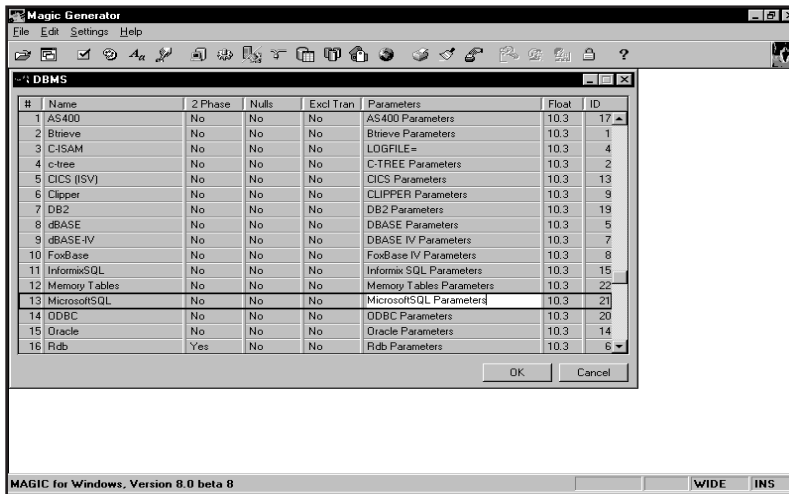


Figure 16-1 The Settings/DBMS Repository

## The [MAGIC\_DATABASES] Settings/Databases Section

Parameters and properties of the Database repository are shown below. Only settings relevant to RDBMSs are covered in this guide.



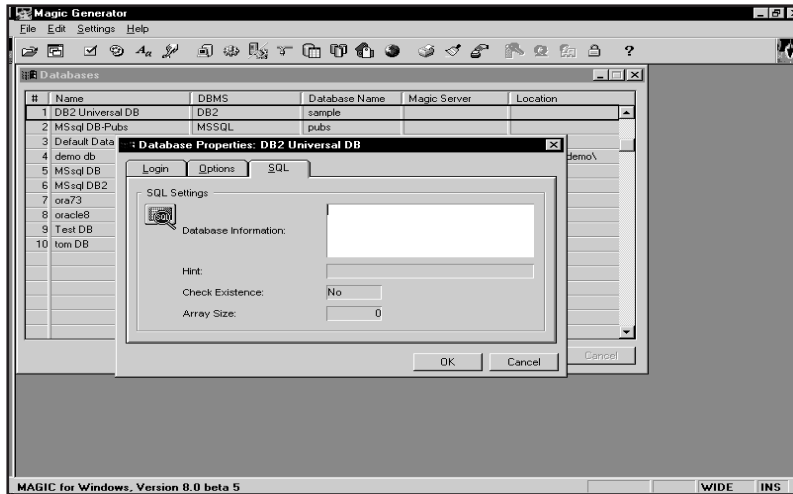


Figure 16-2 Database Repository and Properties Dialog

### **Name**

The name is the name of the database as used in Magic. This can be any name.

### **Dbname**

The dbname is the name of the database as defined by the SQL server. In Oracle this is the ORACLE\_SID.

### **Magic Server**

When using the Magic Client/Server architecture, the database resides on the server, and the server portion of Magic accesses that database. You may select a server name from the available Magic servers as defined above.

When the database is on the same computer, or when the Open Client/Server architecture is used, do not specify the Magic server.

## ***Location***

In most RDBMSs, the database location is transparent to the application. Leave this entry blank with the following exceptions:

- In Rdb, you may specify the location of the database here.
- In INFORMIX SE, the database location is specified, but it is controlled elsewhere by an environment variable.

## ***Database Server***

The name of the SQL server.

In some RDBMSs the server is specified in a connect string. For example:

- MS-SQL and Sybase - always specify the SQL server name
- Oracle - the database alias name in SQL\$NET2 can be specified when using SQL\$NET1. A connect string may be specified such as T:rs6000
- Informix - the Informix server name

## ***User Name and User Password***

Specify the user name and user password when needed. If you want the user name and user password to be dynamic or hidden, you may use a Magic logical name or a secret name, and then initiate the user name and user password with values before connecting to the database.

When these fields have not been filled in, a database login screen appears for the user to fill in the user name and user password. This feature is available starting with Magic Version 7.

## ***Connect String***

The connect string used when connecting to the database is only relevant to Oracle, and writing a string will overwrite the database server definition, username, and password. This parameter provides the benefits of working with SQL\*NET2 specific connection strings.

## ***Magic Locking***

Magic manages its own locking mechanism at the row and table levels without transactions, using the MGLOCK file. These locks are referred to as Magic locks and can be enabled by setting the Magic Locking to Record Lock or Table Lock. Because SQL RDBMS locking is invoked, there is usually no need for an additional locking mechanism. To avoid Magic engine overhead, the use of Magic locks is strongly discouraged.

The recommended setting value is *None*.

**Note:** The Magic Record Lock flag and the File Lock Flag from Magic 7, are combined into a single Magic Locking setting in Magic 8.

## ***Change Tables in Toolkit***

This parameter determines whether a Magic application in Toolkit mode can alter the table structure of any underlying database table.

The recommended value setting is NO.

## ***Database Information***

The Database Information parameter lets you supply database-dependent information that Magic can pass to the underlying DBMS. The use of this parameter is optional. Most of the flags that were specified in Magic 6 and 7 in the database information field are integrated into the toolkit.

## ***Hint***

Some RDBMSs such as Oracle, MS-SQL, and Sybase, allow hinting in the optimizer for processing a query. Magic enables the programmer to enter a hint string that is concatenated to the SELECT statement that builds the Magic dataview. Magic does not evaluate the string and therefore the syntax is the responsibility of the programmer. However, in MS-SQL and Sybase the hint string FORCE\_INDEX can be entered. Magic is then automatically able to use the relevant syntax for forcing the optimizer to use the relevant index. For additional information, please refer to the Technical Information section.

**Note:** It is recommended that you use Hints only in special cases.

## ***Check Existence***

This setting is the same as defined in the DBMS. When creating a new table in the Magic Repository, the property is copied from the Check Existence Flag in the DBMS, and can be overwritten.

## ***Array Size***

The Magic gateways to SQL support array processing. When retrieving rows from the database, the gateway does not retrieve one row at a time, but rather retrieves a group of rows, thus enabling a reduction in network traffic.

Although Magic uses its own array size of rows, these numbers can be overwritten. When scanning a large table, enlarging the array size can enhance performance. It is recommended to use the Magic default. The default should only be changed in special cases.

The array size is copied to the table property only when creating new tables.

## ***External Use in Magic 5.x & Magic 6.x***

Versions of Magic prior to Version 7 and 8 included the External Use parameter in the Database Properties dialog of the *Settings/Databases* repository. For more information see *The Magic Developers Guide and Reference*, for Magic Version 5 or Magic Version 6, Chapter 16, Settings. Setting this parameter to Yes allows non-Magic processes to access the same tables accessed by Magic and therefore automatically adopt the database's locking mechanism.

Magic Version 7 and 8 allows external use by default and introduces a new parameter to change tables in Toolkit mode instead of using the previous External Use parameter.

Setting External Use to YES is equivalent to setting:

1. Change Tables in Toolkit = NO
2. Multi user = YES

The recommended setting value is YES.

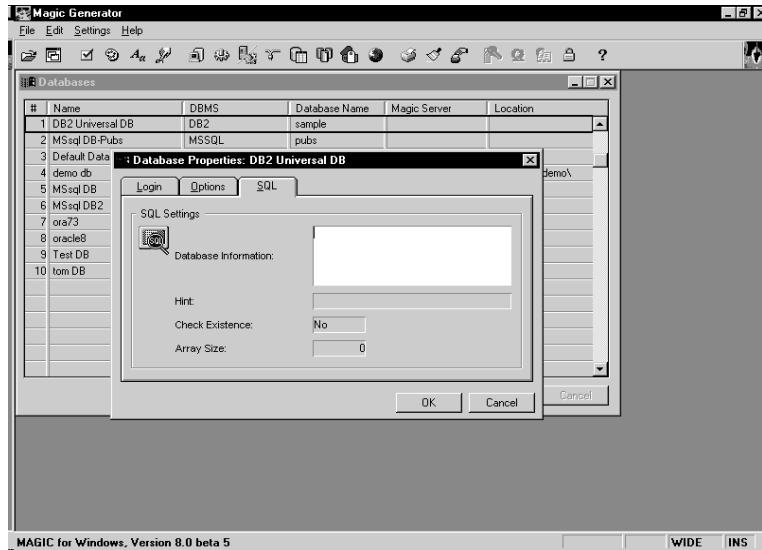


Figure 16-3 Database Properties Dialog

Name	Type	Default	Applicable Gateways
Hint	Alpha 255	N/A	Oracle, Sybase, MS-SQL
Check Existence	Yes/No	No	All
Connect String	Alpha 255	N/A	Oracle
Array Size	Number	20	Oracle, Sybase, MS-SQL, DB2

## ***Multiple Databases***

The definition of a database in the MAGIC.INI file is a logical definition that does not affect the actual SQL database but defines the Magic application's view and behavior.

In the MAGIC.INI file you can define more than one database that in fact points to the same database for various reasons. For example:

- Some of the tables in the database must be accessed by user1 and some by user2. In this case, two identical definitions may be defined for each username.
- For compatibility with older Magic applications, some tables require the use of the Magic Locking mechanism, and some rely only on the RDBMS. This is especially relevant in page-level locking RDBMSs when Magic's row-level locking is required. Therefore, two identical databases can be defined with different settings of the Magic Locks flag.

## ***Multiple Connections***

The SQL gateway supports multiple connections and uses the same connection for all the databases defined in the MAGIC.INI file unless the dbname, User Name and User Password, or Database Server, is different from the existing connection. In this case, the gateway opens a second connection to the database.

**D**atabase meta-data can be defined and maintained using the Magic Table Repository. Every modification to the Magic Table Repository can be synchronized with the underlying database. Magic is limited, by definition, to the manipulation of the data definitions in the repository.

Magic makes no attempt to optimize database structures or to tune database parameters. These tasks should be performed by the database's DBA with external tools. A new table created with Magic is created in a generic form and must be altered later to achieve better performance in a production environment. Magic does not eliminate the need for a good DBA.

## ***Table Repository***

A table defined in the Table Repository is a logical definition known only to Magic programs. In the database this table may be a table, view, synonym, etc. The CREATE TABLE statement includes several parameters relating to the table that does not exist in Magic. You may define several files in Magic that point to the same database object.

The Table Repository entries and parameters are explained below.

### ***Name***

The name that will be used throughout the Magic application. It may be any name.

## ***DB Table***

The actual name of the table as it is defined in the underlying database. This name has the limitations of the specific database. For example, in Oracle the limitations are up to only 30 characters, and begin with a letter.

Since the full name of an object consists of owner.tablename or DATABASE.owner.tablename, Magic takes the owner of the table from the properties of the table.

Unlike previous versions of Magic, when adding a new entry in the Table Repository, Magic 8 copied the name of the table to the DB table column, and replaced blanks with underlines. The table could be overwritten, but not left blank. For example, table name: 'my emp table' will result my\_emp\_table as the DB table name. It is possible to choose a convenient naming convention for all tables.

## ***Database Name***

The DBMS and database entries are taken from the list of databases defined in the MAGIC.INI file.



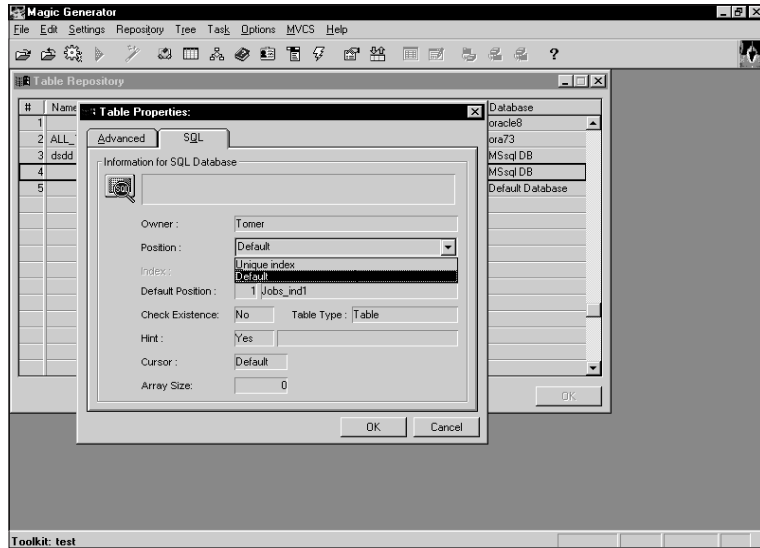


Figure 17-1 Table Properties Dialog

Name	Type	Default	Applicable Gateway
Owner	Alpha	N/A	All
Position	Default/Unique Index/RowID	Default	All
Index	Unique IndexNumber	None	All
Check Existence	Yes/No/Default	Default	All
Table Type	Table/View	Table	All
Hint Flag	Yes/No	Yes	Oracle, Sybase, MS-SQL

<b>Name</b>	<b>Type</b>	<b>Default</b>	<b>Applicable Gateway</b>
Hint	Alpha 255	None	Oracle, Sybase, MS-SQL
Cursor	Default/Yes/No	Default	MS-SQL
Array Size	Numeric 4	N/A	Oracle, Sybase, MS-SQL

## ***Table Properties***

### ***Resident & Cache Strategy***

The Cache strategy and the resident table, determine whether the table should be treated as a resident table or cached. A description of these parameters in the Magic Developer's Guide & Reference Guide. The use of the cache and resident parameters enhances performance because the data is fetched and kept in the memory of the client, and communication with the SQL database is reduced.

## ***Database Information for SQL Databases***

The Database Information parameter lets you supply database-dependent information that Magic can pass to the underlying RDBMS. The use of this parameter is optional. Most of the flags which are specified in Magic 6 and 7 in the database information are integrated into the toolkit.

Most of the flags used here have the SQL\_flagname=Value structure.

## ***Owner***

If the owner of the table is left blank, Magic does not add it to the name of the table. If it is filled, Magic adds the owner to the prefix of the table name. By default, when the Owner is not specified, the Database searches for an object which is owned by the username. It is then used to connect to the database and to the other owners.

## ***Position***

Magic uses its own default as a position key for a table. In Oracle and Informix tables, Magic uses the ROWID column. In other RDBMSs, Magic uses the shortest unique index. This can be overwritten by using another index as the position key. For additional information, please refer to the Unique Identifier section. It is recommended to use the default position.

## ***Index***

When the Magic default is not being used, a unique index can be selected as the position.

## ***Default Position***

This is the index that Magic uses as the default position index. This property is read-only, for the user's information.

## ***Check Existence***

This setting is the same as defined in the DBMS. When creating a new table in the Magic Repository, the table is copied from the check existence flag in the Database, and can be overwritten.

**Note:** If the check existence is set to *No*, tables will not be able to be created in the database. This is because Magic assumes that the table already exists.

In Deployment you should set the Check Existence parameter to *No* to enhance performance.

Setting the Check Existence to Yes, enables Magic to check if the table exists, and if it does not exist, Magic will create it.

### ***Table Type***

The table type can either be *View* or *Table*. If the table type chosen is *View*, then the DBDEL(), and DBCOPY() functions will not work, since a *View* cannot be created by Magic, and therefore cannot be deleted by it.

### ***Hint***

Specifying *Yes* in the hint flag, enables you to fill a hint string. If the hint string is left blank, the hint string specified in the database will be inherited. If the hint flag is not specified, hints will not be used when working with this table.

Some RDMSs such as Oracle, MS-SQL and Sybase allow hinting the optimizer for processing a query. Magic enables the programmer to enter a hint string which will be concatenated to the SELECT statement and builds the Magic dataview. Magic does not evaluate the string, therefore the syntax is the programmer's responsibility. However in MS-SQL and Sybase, the hint string FORCE\_INDEX can be entered. Magic will automatically use the relevant syntax for forcing the optimizer to use the relevant index. For additional information, please refer to the Technical Information section. It is recommended to use Hints only in special cases.

### ***Cursor***

When using the MSSQL gateway, the gateway uses internal DB commands or cursors. Using DB commands instead of cursors, requires separate connections for each result set. Performance is enhanced when the result set is large.

### ***Default***

1. When the table is used as the main task table, Magic uses the cursors.
  2. When the table is used as a linked table, Magic uses DB commands.
- Yes - When this table is used, Magic always uses cursors.

No - When this table is used, Magic always uses DB commands.

Note: When there is a BLOB column in a table, Magic will use the DB commands.

For additional information, please refer to the technical information section.

### ***Array Size***

The Magic Gateways to the SQL support array processing. When fetching records from the Database, the gateway does not fetch one record at a time, but rather fetches a group of records, thus enabling a reduction in network traffic.

By default, the array size is zero, which means that Magic uses its own array size of records, and this number can be overwritten. However, when scanning a large table, enlarging the array size can enhance performance. It is recommended to use the Magic default. Changing the array size in the table will overwrite the database properties setting.

### ***DB Table***

The actual name of the table as it is defined in the underlying database. This name has the limitations of the specific database.

The full name of an object consists of owner.tablename, or DATABASE.owner.tablename. The full name DATABASE.owner.tablename may not be specified here. The owner can only be specified in the owner property when you create a new owner, otherwise the owner is taken from the connect. In Get Definition, you can specify a full name.

In earlier versions of Magic, if left blank, Magic uses its default naming XXFILL099, where 99 is the number of the row in the table repository. In Magic 8, a name must be entered, and it must not begin with a blank.

You may choose a convenient naming convention for all tables.

## ***DBMS and Database***

The DBMS and database entries are taken from the list of databases defined in the MAGIC.INI file.

## ***Table Properties***

### ***Resident & Cache Strategy***

The Table Properties parameters let you determine whether the table should be treated as a resident table or cached. You can find a description of these parameters in *The Magic Developer's Guide*.

The use of the Table Properties parameters enhances performance because the data is fetched and kept in the memory of the client, and communication with the SQL database is reduced.

## Column Definition and Properties

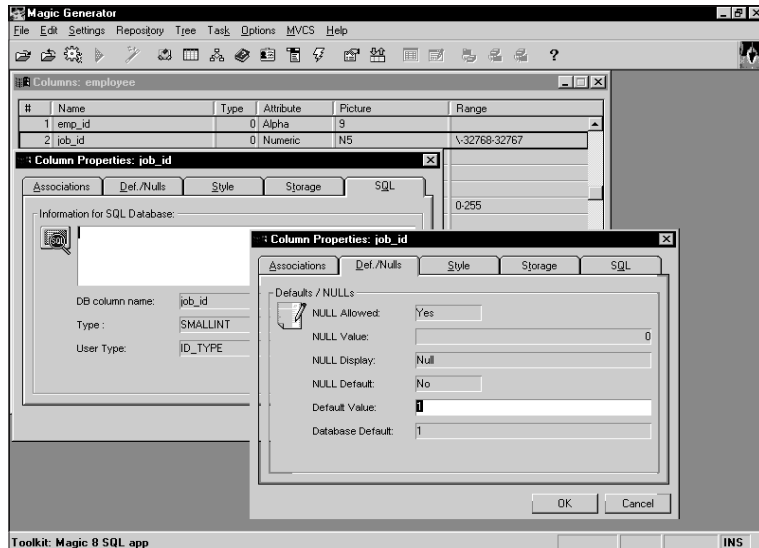


Figure 17-2 Column Properties Dialog

The definitions of the column and its properties are logical and relevant only to Magic. It is important to understand the mapping of the Magic attribute to the underlying RDBMS data type.

### **Name**

The name that will be used throughout the Magic application. This may be any name.

### **Attribute**

The Attribute column is where you specify the Magic attribute.

In future versions of Magic the attribute of the field will be derived from the underlying RDBMS, and only the relevant data types will be displayed.

## ***Properties***

Several settings are relevant to RDBMSs.

### ***Null Allowed***

The Null Allowed field is where you determine whether to allow null to the column or not. When creating a new table, the null constraint is taken from here. The default is taken from the DBMS section in the MAGIC.INI file.

In Runtime this setting is relevant only to Magic. The Null Allowed setting tells Magic whether to allow nulls to be inserted into the field before being passed to the database.

### ***Null Value and Display***

The null value for calculation and the display string that is displayed when the value of the column is null. For additional information, refer to the explanation of null settings in the application properties in *The Magic Guide to Application Development* and in *The Magic Reference*.

### ***Null Default and Default Value***

This is the value which should be inserted into the database when selecting the column, however it is not updated. Please refer to the Default/Null setting in the application properties, in the Magic Developer's Guide.

### ***Database Default***

This is the default value for the column in the database. If a program creates records and does not select this column, the default value will be assigned to the column by the RDBMS.

When loading the table definition, Magic will load the default. If the default is a constant, Magic will also load it into the default value property to serve as a default for the column inside Magic. When creating a new table, Magic will add to the CREATE TABLE statement this string as the default value for that column. For example, when creating a database default 'defvalue' in MS-SQL



table, Magic will generate the following: CREATE TABLE owner1.table1 (Col1 CHAR (5) NOT NULL DEFAULT 'defvalue').

**Note:** Magic adds this string as is, to the CREATE statement. It is the responsibility of the developer to add quotes or format it according to the datatype of the column.

### ***Stored As***

Every attribute in Magic has several storage types. If not specified otherwise, the SQL gateway uses the most appropriate default mapping to the underlying RDBMS data type. For example, the Alpha attribute may be stored as a Zstring, in Magic. This storage is internal to Magic only.

When switching from one RDBMS to another, Magic tries to use the same storage type. However, for compatibility reasons, the storage type may be different from one RDBMS to another.

Do not change the Magic default storage types. Please refer to the Mapping Datatypes section for more information.

### ***DB Column Name***

This is the actual name of the column as it is defined in the underlying database. This name has the limitations of the specific database.

Unlike previous versions of Magic, when adding a new column to a table, Magic 8 copied the name of the column to the DB name column, and replaced blanks with underscores. It is possible to overwrite it, but you are unable to leave it blank. For example, column name: 'emp id' will result emp\_id as the DB column name. It is possible to choose a convenient naming convention for all columns.

### ***Information***

The Information parameter lets you supply database-dependent information that Magic can pass to the underlying DBMS. The use of this parameter is optional. Most of the flags which are specified in Magic 6 and 7 in the database information are integrated into the toolkit.

## ***Type***

This is the datatype of the column in the underlying RDBMS. It is loaded when the table definition is loaded. When creating a new table in the Table Repository, Magic uses its own default mapping and there is no need to specify the database datatype. In order to force the use of a specific data type in cases where the Magic default is not sufficient, a different datatype may be specified.

For example by default, the Magic date column is mapped to the 'DATE' datatype in Oracle. However, in order to force Magic to map the date column to a different datatype, specify CHAR (8), as the SQL type. Please refer to the Mapping datatype section for a complete list of valid and default mapping between Magic and each RDBMS.

## ***User Type***

This is the User Defined Type (UDT) as defined in the database. In most RDBMS, a user defined type can be created to redefine a system datatype. For example, the UDT 'description' is based on 'VARCHAR(20)'. Once a UDT is created, it can be used in the CREATE TABLE and ALTER TABLE statements, as well as attaching defaults and rules to it. When loading a table definition, Magic loads the UDT of each column, and the system datatype which it is based upon. This is for internal use only. Currently, when creating tables via Magic, Magic uses only the SQL type.

## ***Database Information***

The Database Information parameter lets you supply database-dependent information for Magic to pass to the underlying RDBMS

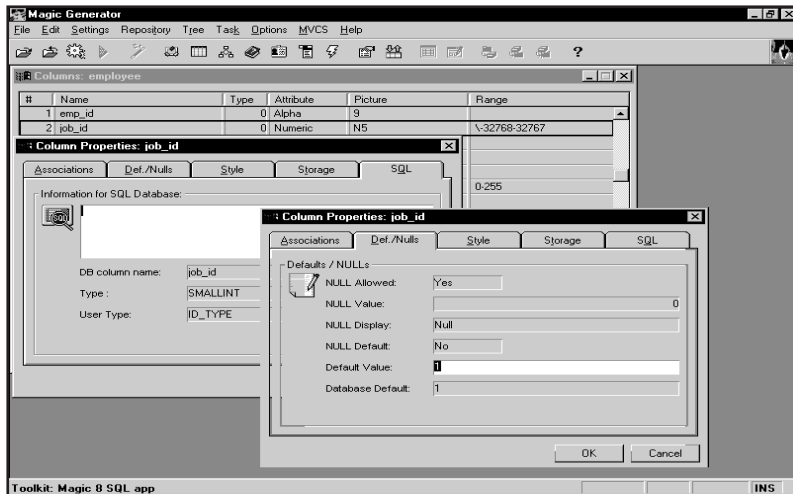


Figure 17-3 Database Information Field

# Index Definition and Properties

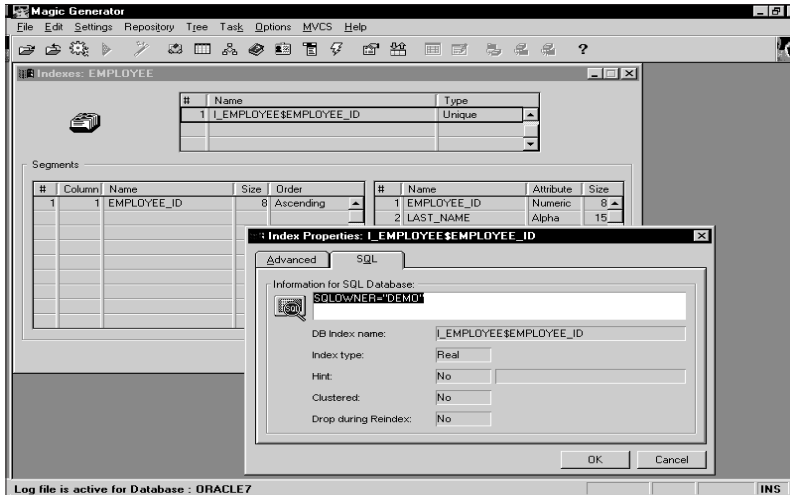


Figure 17-4 Index Properties Dialog

The definition of the index is logical and relevant only to Magic. The indexes in the database may be different from the index defined in Magic. By using the Get Definition option, you may get the actual index structure in the database.

The main idea is to build an index in Magic that matches an index in the database so that no sort operation is required by the optimizer, resulting in a fast response time.

Defining an index in Magic has two results:

- When creating a new table, if the index type is real, Magic creates an index in the database to match this index.
- In Runtime, the Magic index determines how the ORDER BY clause is added to the SELECT statement issued by Magic. The segments specified in the index are the columns that are specified in the ORDER BY clause.

For example, a unique index with the two segments Deptno Asc and Ename Desc causes Magic to generate:

```
SELECT ... FROM ... WHERE ... ORDER BY Deptno  
Asc, Ename Desc'
```

### ***Name***

The name that will be used throughout the Magic application. It may be any name.

### ***Type***

The Type controls whether the index is unique or not.

Some SQL gateways require at least one unique index to work with a table.

It is advisable to make all your indexes unique because a unique index is preferable to a non-unique index.

### ***Segments***

Add the segments of the index from the Magic Column list.

The order can be ascending or descending. When you want to create an index in the database, you must be careful because some of the databases support only ascending indexes. Refer to the appropriate RDBMS documentation to verify this issue.

## ***Properties***

### ***Direction***

The Direction property determines which behavior will be used when moving back and forth over a table in an online task. Performance problems might occur when using two-way indexes with databases that do not traverse throughout their indexes in both directions, such as Informix.

## ***Range Mode***

Not relevant in SQL.

## ***Information***

The Information parameter lets you supply database-dependent information that Magic can pass to the underlying DBMS. The use of this parameter is optional.

Most of the flags which were specified in Magic 6 and 7 in the database information, are integrated into the toolkit.

## ***DB Index Name***

This is the actual name of the index, as it is defined in the underlying database. This name has the limitations of the specific database.

Unlike previous versions of Magic, when adding a new index to a table, Magic 8 copied the name of the index to the DB Index Name, and replaced blanks with underscores. It is possible to overwrite the index Name, however it must not be left blank.

For example, Index name: 'emp ind1' will result emp\_ind1 as the DB Index name.

## ***Index Type***

The Index Type property, tells Magic whether the Index is contained in the database or defined only in Magic. When creating a new table, if the indexes need to be created according to the guidelines of Magic, the index type should be specified as real.

This flag has no effect on existing tables in runtime. For example, when a view is accessed, a virtual index needs to be added. Virtual Indexes are not recommended instead of sorting in the program, now that we have Sort Using RDBMS. This enables the records to be fetched in the required order and sorting is not necessary in Magic.

## ***Clustered***

In MS-SQL/Sybase/Informix, this flag controls whether the index will be clustered when tables are created via Magic. In Magic 6/7, the first unique index is created as clustered, by default. In Magic 8, the developer decides which index is clustered, otherwise the table is created without any Clustered Index. It might also cause performance problems if it is defined on a table with a high insert ratio.

A Clustered Index refers to the physical data which is stored in the order of the index. A Clustered Index is efficient for scanning sets of data in the order of the index, and less efficient when trying to access one of the records directly, using that index. For additional information, refer to the MS-SQL/Sybase SQL reference.

## ***Drop During Reindex***

Controls whether Magic will drop the Indexes when the Reindex option is used.

## ***Get Definition***

When using an existing database, it is often preferable to read the existing table descriptions for the RDBMS data dictionary tables than to redefine those descriptions in Magic. This reduces the risk of error and makes the development process quicker and easier.

The Get Definition option is available from the option menu The Get Definition utility lets the gateway user access and modify existing database definitions immediately.

You can load either a single file definition or multiple files. The menu will be active if an entry in the repository has been assigned to one of the SQL databases.

To load a single table:

1. Access the Table Repository.

2. Place the cursor on a new line.
3. Choose the database.
4. Write the name of the table to be loaded.
5. Click the Options menu. The Get Definition option is enabled.
6. Click on the Get Definition option. The Loading Window is displayed. The Loading Window closes automatically. The table now has the appropriate fields.
7. Enter the Columns field and Zoom to see if the columns have been loaded.
8. Enter the index field and Zoom to see that the database indexes have been loaded as Magic indexes.

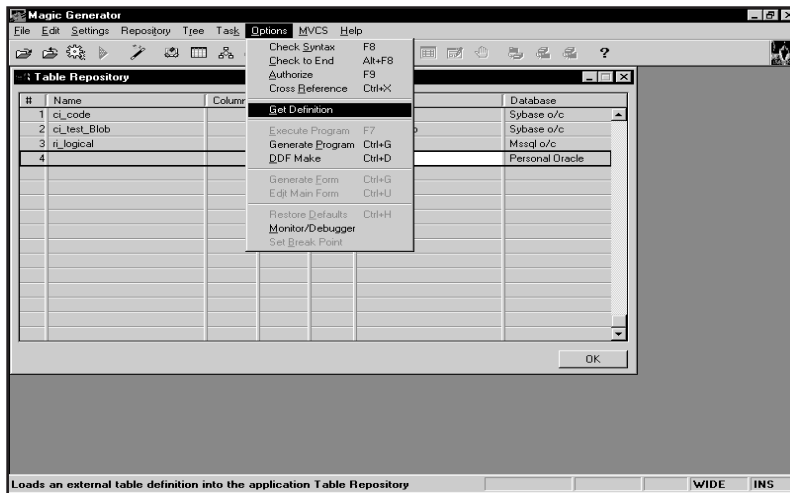


Figure 17-5 Table Repository, Get Definition Enable

Click on the Get Definition option.

The Loading window shown in Figure 17- is displayed.



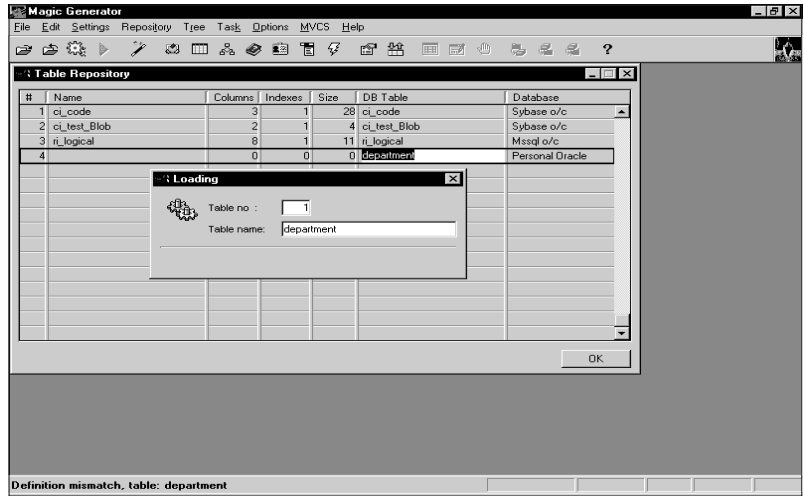


Figure 17-6 Table Repository, Loading Window Open

The Loading Window closes automatically. The table now has the appropriate fields.

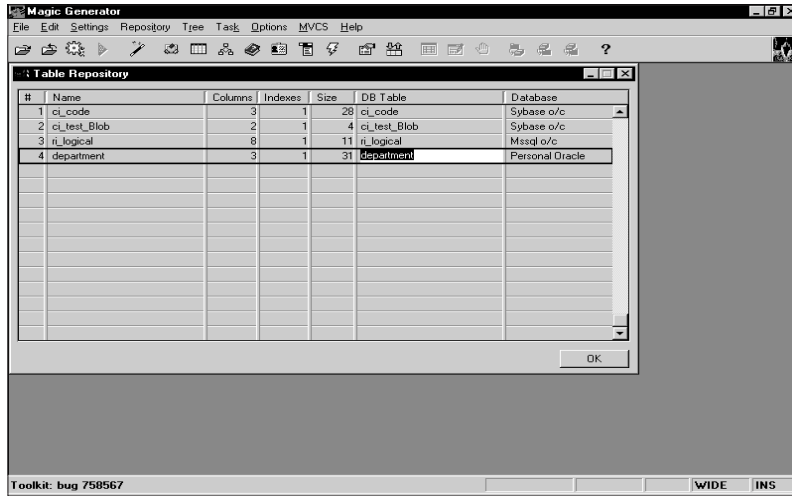


Figure 17-7 Table Repository, Loading Window Closed

To load several tables at once:

1. Place the cursor on the title line.
2. Select Get Definition from the Options menu.
3. The Load definition window appears.
4. Enter the Database field and Zoom.
5. Choose the database you want to load from.
6. Enter the Tag Files field and Zoom.

Select Several, if you want to load all the database tables accessible to the database, as defined in the Database Table with user name and password, or specify multiple tables.

The Table Selection window opens.

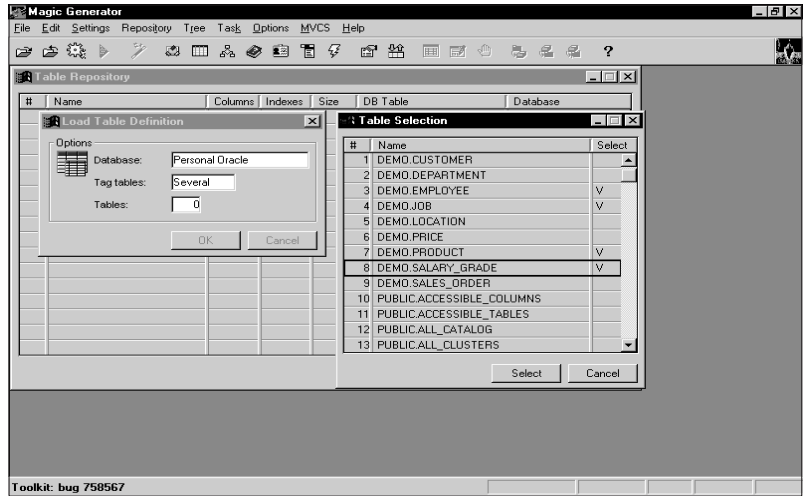


Figure 17-8 Table Selection Dialog

To tag the requested files:

1. Highlight the desired items.
2. Press the space bar. A check appears in the Select field to show that the table has been selected for loading.
3. Find the Order Lines table and press the space bar.
4. Find the Customers table and press the space bar.
5. Find the Userids table and press the space bar.
6. Find the Counters table and press the space bar.

## Notes on Get Definition

Note the following about Get Definition behavior:

- When loading existing table definitions, Magic uses the most appropriate data type for each database column.

- Magic matches special types if supported.
- The Load File Definition option fails if Magic encounters an unsupported data type.
- There are specific differences in the Load File Definition capabilities for each database.

Oracle/Rdb will not show views that do not return DBKEYS in the Table Selection window.

## ***View Definition Loading***

SQL also offers view definitions, which can be accessed in the same way as the table definitions are accessed. An SQL view is a virtual table defined by a query. An SQL view is accessible as a table but does not physically contain rows. You can access an SQL view in query mode, and in some cases depending on the RDBMS, update them just as if they were tables in the RDBMS.

A view does not contain an Index. Therefore when loading a view, the developer **MUST** define a virtual unique index. This is because Magic must have a position for the Table.

In RDBMSs, such as Oracle, Informix or Rdb, which have a ROWID/DBKEY, if the view is based on one Table, the ROWID can be obtained, and the position can be ROWID. Otherwise, the Virtual Unique Index is necessary as the position index.

## ***Table Modification***

When a table is altered In Magic, Magic converts the data, renames the table, and recreates the table with the new properties. Magic also recreates indexes that have been changed. These changes can be made in two ways:

- Magic can implement the changes using the RDBMS DDL statements, such as ALTER, RENAME, and DROP.

- Magic can approximate what is done in ISAM files by creating a temporary table, copying the existing data to the temporary table, creating a new table, and copying the data into the new table.

For performance reasons, it is often better to let the DBA do the conversions using the RDBMS DDL tools than to have Magic fully convert the table. In fact, any type of ALTER command that can be implemented by the RDBMS should be done by the DBA.

- The Oracle gateway does not use its own ALTER command. If columns exist in the table that do not appear in Magic's Table Repository, the columns will be erased by a conversion.
- The Oracle/Rdb gateway does everything using DDL commands, except for renaming tables and columns.
- The Informix gateway does not use its own ALTER command. If there are columns in the table that do not appear in Magic's Table Repository, those columns will be deleted by a conversion.

When changing Tables, be aware of the Change Tables in the Toolkit setting and how that will affect the database conversion.

Magic alterations in index sequential files usually demand greater system resources than in SQL. For instance, a new index in SQL does not rebuild the data as it might in ISAM files. Some RDBMSs can remove fields without rebuilding the data. Only a change in the data repository is necessary. Many data type alterations can be done in the same way.

[This page intentionally left blank]

---

**M**apping RDBMS data types to Magic attributes is usually quite simple. However, programmer intervention may be needed to explicitly specify the desired data type in the RDBMS.

## ***Magic Default Mapping***

When you create a new column in Magic you may choose its attributes, such as Alpha, Numeric, Logical, etc. Every SQL gateway contains a table that determines the default data types of the underlying RDBMS to be used for each attribute. This data type is used when creating the table in the database, and to convert the data from the RDBMS to the Magic application in Runtime.

Each attribute may have several storage types. The gateway decides which data type to use according to the attribute and the storage type.

Some storage types are irrelevant and should be avoided when using SQL. For example, the use of the Numeric attribute, stored as MS Float Basic, causes Magic to convert the data from the RDBMS to Magic and back again to the RDBMS. This results in poor performance.

Use only the attribute's default storage type. Some default mapping has changed from Magic 7 to Magic 8 and might require a small change in the application:

- Date string is of length 8 only. The application should be adjusted but no change is required in the database, but if you are using CHAR (6) as the SQLTYPE in Magic 6 or Magic 7, you should change your data.
- Informix default storage for the Alpha is Zstring, instead of String in Magic 6 and Magic 7. No change in the application is required.
- Informix default storage for Time is String Time, instead of Integer Time in Magic 6 and Magic 7. No change in the application is required.
- Informix default SQL type String Time is 'DATETIME HOUR TO SECOND' instead of CHAR (8). In case STRING TIME is mapped to CHAR in the database in Magic 6 and Magic 7 applications, simply add the SQL type CHAR (8), to that column.

## ***Get Table Definition Mapping***

When performing the Get Table Definition Mapping option, the SQL gateway reads the information on the data types of each column in the table to be loaded from the system tables in the SQL database. This ensures that the SQL gateway uses the most appropriate attribute and storage type for this RDBMS data type. Magic also reads the length, precision, and scale of the columns to fill in the picture field in Magic. However, in some cases the information in the database is not sufficient, and programmer intervention is required. Some examples are listed below:

- Longraw, Image, and Text data types have no length, so the picture field is left empty to be filled in by the programmer.
- Longraw, Image, and Text may be used in Magic for Memo and Blob fields. When performing the Get Table Definition operation, the gateway uses one of these as a default, and a change may be necessary to use the correct one.

In Sybase and MS-SQL, binary data types may be used for several attributes. When performing the Get Table Definition operation, the gateway uses a default attribute.



- In Oracle all numeric columns are internally stored as 40 digit precision, so a Get Table Definition operation will give the right picture, but the attribute will always be Float. The programmer can change the attribute to another attribute, such as signed integer.

Refer to the RDBMS to Magic Mapping tables for each RDBMS.

## ***Enforcing a Data Type - SQL TYPE***

When you want to use a different data type than the default, you can explicitly specify that data type in the Type parameter on the SQL tab in the Properties dialog of the column.

For example, a one-character alpha column may be better stored as CHAR(1) than as VARCHAR2(1). You may want to specify CHAR(1) to enforce that data type, because the SQL gateway default is VARCHAR2(1). Please note that the use of a data type that does not fit the attribute in Magic may be harmful.

The SQL TYPE may also be filled in by the SQL gateway when performing the Get Table Definition operation.

## ***Magic Data Types***

The following table lists Magic attributes with the supported storage types, and the valid data types in each RDBMS. Each entry in the table has, for each RDBMS, a default data type that can be forced by specifying that type in the Type parameter on the SQL tab in the Column properties dialog.

The table summarizes how Magic data types map to MS-SQL, Oracle, ODBC, Sybase, DB2, and Informix data types.

<b>Magic Data Type</b>	<b>MS-SQL Data Type</b>	<b>Oracle Date Type</b>	<b>ODBC Data Type</b>	<b>Sybase Data Type</b>	<b>DB2 Data Type</b>	<b>Informix Data Type</b>
Alpha Zstring	Char/Text *	Varchar2/ Long/ Char	sql_char/ sql_longvarchar*	Char/Text *	Char/long/ varbinary*	Char
Numeric Signed Integer	Smallint, Integer	Number	sql_tinyint/ sql_smallint	Smallint, Integer	Tinyint/Smallint/ Integer	Smallint, Integer
Unsigned Integer	Binary	Number	sql_integer/ sql_binary sql_float	Binary	Tinyint/Smallint/ Integer	Char
Numeric Float	Float, Real	Number	sql_float/ sql_double*	Float, Real	Float/Double Decimal	Float
String Date	Datetime/ Char **	Date/Char /Raw**	sql_date	Datetime/ Char**	Date/Char**	Date (yyyymmdd) /Char**
String Time	Char	Char/Raw /Date	sql_time	Char	Time	Date time Hour to second/char
String Memo	Binary /Image	Raw/ Longraw*	sql_binary	Binary/ Image	Char/ varchar/ longvarchar (for bit data)	Varchar/Text /Char**
Logical Integer	Bit/ Smallint	Number	sql_bit	Bit	Char (for bit data)	Small integer
String Logical	Binary	Raw	sql_binary	Binary	Char (for bit data)	Char
Blob	Image	Longraw	sql_longvarbinary	Image	Blob	Byte
All others	Binary/ Image	Raw/ Longraw	sql_binary/ sql_long varbinary	Binary/ Image	Char (for bit data)	Char

Note: The first data type is the default.

\*- The supplied data type is determined by the length of the column

\*\* -If you want to map Magic String Date to (fill in database) character, specify char (8) as the SQL type.

## ***Date and Time Mapping***

The default mapping of the date attribute in Magic is Date or Time in the underlying RDBMS. However, two issues must be taken into consideration:

1. Most RDBMSs store the date and the time in the same column, while Magic has a separate attribute for Date and for Time.
2. A date value of zero, as used in many Magic applications written for ISAM files such as Btrieve, is illegal in all RDBMSs.
3. The complete Mapping Table also contains information on Date and Time mapping.

### ***Defining Date Columns in Magic***

Three attributes are important when defining a Date in Magic.

#### ***Size***

When defining a Date attribute in Magic through Version 7, the default size is 6, and it could be changed to 8. In Magic 8, the size is changed to 8, so the whole year portion will be stored and handled in the database and not by Magic. This may be helpful when dealing with the year 2000.

#### ***Stored As***

The default stored type and the recommended stored type is String Date. The String Date storage type does not require internal conversion of the data, and if needed, lets you map the date to CHARACTER in the database.

## **SQL TYPE**

If you are not required to use a zero date, you should use the Underlying Date data type. This is the Date in Oracle, Informix, and DB2, and the Datetime in MS-SQL and Sybase. To overcome the problem of date columns that do not have any values, you can:

- Allow NULLS for these columns.
- Use the INIT in the SELECT REAL command to initialize the column with a base date.
- Use the Magic default with a base date.
- Use the database default in case they are not selected in a create program

If you must use a zero date, you can map the date attribute to CHARACTER data type in the database. This can be done by specifying the SQL TYPE to be:

CHAR (8)

In this case, the column is stored as CHARACTER, and you may enter a zero value.

## ***Mapping on Existing Data***

- Mapping as Date attribute - Mapping a Magic Date attribute to Date or Datetime data types causes the time portion to be invisible to the Magic application. When inserting or updating these columns, the SQL gateway adds a constant time, usually 24:00:00, to the date entered by the user. Otherwise the RDBMS uses its defaults, and ranging on these columns does not cause any problems.
- Mapping as Alpha attribute - If the Time portion has to be seen by Magic, you may map the Date or Datetime columns to an alpha column in Magic with a length of 19, and specify the SQL TYPE to be: "DATE" or "DATETIME" in the SQL properties of the column. Then the full date, including the time portion, will be brought to Magic as a string in the

database format, and no Date or Time checks will be performed on this string.

To enforce these checks you can create two virtual fields, Time and Date, using the DVAL and TVAL functions to map these fields to the real column.

A different flag in each SQL gateway can be turned on to map Date or Datetime columns to alpha fields in Magic when performing the Get Table Definition operation.

## ***TIME Attribute***

When using TIME as the attribute, the Magic default is usually acceptable. In some RDBMSs, such as DB2 and Informix, a TIME (or DATETIME HOUR TO SECOND) data type exists. In other RDBMSs a character is used so that the TIME column can be viewed from other tools as well.

## ***Magic Equivalents for the RDBMS Data Types***

The following tables are Magic equivalents for the RDBMS data types.

## ***Informix Data Types***

---

### **Magic Equivalents for Informix Data Types**

---

<b>Informix Data Type</b>	<b>Attribute</b>	<b>Magic Storage Type</b>	<b>Storage Size</b>
char(n)	Alpha	Zstring	1-32700
varchar(n)	Alpha	Zstring	1-255
integer	Numeric	Signed int	4
smallint	Numeric	Signed int	2
float	Numeric	Float	8
smallfloat	Numeric	Float	4
money(p,s)	Numeric	Float	8
decimal(p,s)	Numeric	Float	8
date	Date	String date	8
serial	Numeric	Signed int	4
date-time	Alpha	Zstring	1-25
interval	Alpha	Zstring	1-24

---

# Sybase Data Types

## Magic Equivalentents for Sybase Data Types

Sybase Data Type	Attribute	Magic Storage Type	Storage Size
Char(n), Varchar(n)	Alpha	Zstring	n, 1-255
Text	Alpha	Zstring	11
Int	Numeric	Signed Integer	4
Smallint	Numeric	Signed Integer	2
Tinyint	Numeric	Unsigned Integer	1
Numeric(p,s)	Numeric	Float	8
Decimal(p,s)	Numeric	Float	8
Float(8)(double precision)	Numeric	Float	8
Float(4)(real)	Numeric	Float	4
Money	Numeric	Float	8
Smallmoney	Numeric	Float	4
Datetime*	Alpha/Date	Zstring/String Date	23/8
Smalldatetime*	Alpha/Date	Zstring/String Date	16/8
Binary(n), Varbinary(n)	Alpha	Zstring	n, 1-255
Image	Blob	Binary Large Object	12
Bit	Logical	Integer Logical	1
Timestamp	Alpha	Zstring	8

\* - By default, Sybase Date types (datetime, smalldatetime) are mapped to Magic date storage type. If you want to be able to see all parts of a datetime/smalldatetime column in the format YYYY/MM/DD HH:MM:SS.mmm for datetime, and YYYY/MM/DD HH:MM for smalldatetime, you should map the Sybase Datetime/Smalldatetime to Alpha. This can be done by specifying SQL\_DATETOALPHA=Y in the Database

Information field in the Database Properties dialog (see figure 3-2 in chapter 3).

## Oracle Data Types

The following table shows the results of a Magic Get Definition operation from an Oracle Table.

Magic Equivalentents for Oracle Database Data Types					
Oracle Data Type	Attribute	Magic Storage Type	Storage Size	SQL Type	Picture
Varchar2(n)	Alpha	Zstring	1-2000		n
Char(n)	Alpha	Zstring	1-255	CHAR(n)	n
Long	Alpha	Zstring	0*	LONG	0
Raw(n)	Alpha	Zstring	1-255		n
Long raw	Alpha	Blob	0*	LONGRAW	0
Number	Numeric	Float	8		10.3
Number(p,q)	Numeric	Float	8		p-q.q
Number(p)	Numeric	Float	8		p
Date**	Date	String Date	8		DD/MM/YYYY
Rowid	Alpha	Zstring	19		18

\* -You must set the Storage size for Long and Long raw to the appropriate size for your application.

\*\* - If you specified the DBMS flag SYB\_DATETODATE is used to tell the gateway to map a STRING\_DATE to a sybase datetime datatype.



# SQL Server Data Types

## Magic Equivalentents for Microsoft SQL Server Data Types

SQL Server Data Type	Attribute	Magic Storage Type	Storage Size
Char(n), Varchar(n)	Alpha	Zstring	n, 1-255
Text	Alpha	Zstring	Default
Int	Numeric	Signed Integer	4
Smallint	Numeric	Signed Integer	2
Tinyint	Numeric	Unsigned Integer	1
Numeric(p,s)	Numeric	Float	8
Decimal(p,s)	Numeric	Float	8
Float(8)(double precision)	Numeric	Float	8
Float(4)(real)	Numeric	Float	4
Money	Numeric	Float	8
Smallmoney	Numeric	Float	4
Datetime*	Alpha/Date	Zstring/String Date	23/8
Smalldatetime*	Alpha/Date	Zstring/String Date	16/8
Binary(n), Varbinary(n)	Alpha	Zstring	n, 1-255
Image	Alpha	Zstring	Default
Bit	Logical	Integer Logical	1
Timestamp	Alpha	Zstring	8

\* - By default, Microsoft SQL Server Date types (datetime, smalldatetime) are mapped to Magic date storage type. If you want to be able to see all parts of a datetime/smalldatetime column in the format YYYY/MM/DD HH:MM:SS.mmm for datetime, and YYYY/MM/DD HH:MM for smalldatetime, you should map the SQL Server Datetime/Smalldatetime to Alpha. This can be done by specifying SQL\_DATETOALPHA=Y in the Database Information field in the Database Properties dialog (see figure 3-2 in chapter 3).

## DB2 Data Types

---

### Magic Equivalents for DB2 SQL Data Types

---

DB2 Data Type	Attribute	Magic Storage Type	Storage Size
char(n)	Alpha	Zstring	1-254
varchar	Alpha	Zstring	1-4000
longvarchar	Alpha	Zstring	32700
decimal(p.5)	Numeric	Float	8
smallint	Numeric	Signed Integer	2
integer	Numeric	Signed Integer	4
float	Numeric	Float	4
double	Numeric	Float	8
blob	Blob	Blob	
date	Date	String Date	8
time	Time	String Time	6
timestamp	Date	String Date	8

---

# ODBC Data Types

---

## Magic Equivalents for ODBC SQL Data Types

---

ODBC Data Type	Attribute	Magic Storage Type	Storage Size
SQL_CHAR	Alpha	Zstring	1-255
SQL_VARCHAR	Alpha	Zstring	1-255
SQL_LONGVARCHAR	Alpha	Zstring	Default=0
SQL_DECIMAL	Numeric	Float	8
SQL_NUMERIC	Numeric	Float	8
SQL_SMALLINT	Numeric	Signed Integer	2
SQL_INTEGER	Numeric	Signed Integer	4
SQL_REAL	Numeric	Float	4
SQL_FLOAT	Numeric	Float	4
SQL_DOUBLE	Numeric	Float	8
SQL_BIT	Logical	Integer Logical	1
SQL_TINYINT	Numeric	Unsigned Integer	1
SQL_BIGINT	Numeric	Signed Integer	4
SQL_BINARY	Alpha	ZString	1-255
SQL_VARBINARY	Alpha	ZString	1-255
SQL_LONGVARBINARY	Blob	Blob	
SQL_DATE	Date	String Date	8
SQL_TIME	Time	String Time	6
SQL_TIMESTAMP	Date	String Date	8

---

## Data Definition Rules

Characteristic	MS-SQL	Oracle	Sybase	Informix Online	Informix Standard Engine	DB2
Longest object name (user, column, table, view, index)	30	30	30	18	18	18
Maximum length of char/binary	255		255	255	255	254
Maximum length of varchar				32767	32767	4000
Maximum length of longvarchar						32700
Most columns in a table, view	250	254	250			255
Maximum length of row (row length)	1962	32511	1962	32700	32511	4005
Maximum index length	256	255	256	255	120	
Most columns in an index	16		16			16
Maximum number of Blob columns	10	7	10	20	N/A	7

Characteristic	MS-SQL	Oracle	Sybase	Informix Online	Informix Standard Engine	DB2
Maximum size of each Blob field	2GB	2GB	2GB			
Maximum number of segments per index		16		16	8	
Maximum key/index expression length		255				255
Maximum alpha length		2000		32700	32511	
Maximum Memo field length		32767 bytes		32700	32700	
Maximum number of memo fields supported		1*				
Maximum number of indexes				100(inf 5) 77 (inf 7)	100	

\* Magic's Table Checker cannot distinguish between Oracle Long and Long raw and Magic Memo fields. If Magic finds more than one field of any of these types, it will return the following error message: "Database supports one memo field in record".

[This page intentionally left blank]

# *Understanding Magic & SQL Behavior*

# 19

---

**T**he Magic engine uses SQL language internally to communicate with SQL databases. This chapter discusses the Magic engine's internal functions and requirements when working with RDBMSs.

## *Unique Identifier*

Each row must have a unique identifier for Magic to perform a table operation such as browse, locate, update, etc. A column or combination of columns is added to the SELECT statement and kept inside Magic to uniquely identify each record, known as a row in SQL, retrieved by Magic. These column values are used afterwards to get the row's most updated status when the row is retrieved again from the database.

## *Unique Identifiers in RDBMSs*

Some RDBMSs contain a hidden column for each row in a table. The hidden column receives a value that is the row's ID from the time the row is created until the time the row is deleted. This column cannot be updated by the user. The RDBMSs containing the hidden column include:

- Informix-ROWID
- Oracle-ROWID
- Rdb-DBKEY

Some RDBMSs, including MS-SQL, Sybase, and DB2, do not contain such columns, or else they contain a column called `TIMESTAMP`. Unlike the `ROWID`, `TIMESTAMP` is optional, and you must declare it when creating a table. `TIMESTAMP` changes its value automatically every time the row is updated.

## ***Unique Identifier in Magic***

When `ROWID` and `DBKEY` can be used, as with tables in Oracle, Informix, and Rdb, Magic adds a unique identifier to the `SELECT` statement to use in the `WHERE` clause in update and delete commands.

Example:

```
SELECT empno,ename,rowid
FROM emp
ORDER BY empno asc
```

When the user updates a row, the following command is issued:

```
UPDATE emp SET ename='nina' WHERE ROWID=value
```

where `ROWID` contains the value that was retrieved and kept.

The Position Index will be used when `ROWID` or `DBKEY` cannot be used in the following cases:

- There are views on more than one table in Oracle, Informix, and Rdb, and do not have `ROWID/DBKEY`.
- There are tables and views in Sybase, MS-SQL, and DB2 that do not have a `TIMESTAMP`.
- MagicGate to ODBC is installed.

If a `TIMESTAMP` was defined in the table and can be used, Magic performs in the same way as it performs with the `ROWID` with one exception. The row is retrieved again after each update because the `TIMESTAMP` has been changed. This results in poor performance.



## ***Position Index***

The default position as defined in the properties of the table, causes Magic to use the shortest unique index defined for a table or ROWID column where applicable. Forcing a specific index to be used as the position by Magic can be done by selecting a unique index as the Position Index.

Magic uses the Position Index to specify each specific row, when there is no unique identifier such as ROWID or DBKEY. The Position Index columns are added to the SELECT statement and consist of the WHERE clause in the UPDATE and DELETE statements.

Because the Position Index is used quite frequently, it should be indexed where appropriate.

## ***Unique and Non-Unique Indexes***

Magic performs an internal function called START POSITION to reposition a row in the screen after it has been updated, or to change an index and reorder the screen, or when a cursor is closed due to a Commit transaction. For that function to work properly, the rows on the screen must be sequentially and uniquely ordered.

Therefore, for each SELECT statement except those that range on an exact value of the ROWID or Position Index, an ORDER BY clause is added. This ORDER BY clause consists of all the index segments defined in the program, with the following exception: If the index is non-unique, the ROWID or DBKEY, or the segments of the Position Index, are appended to the ORDER BY clause to make it unique.

For example, in Oracle, when the table named *emp* is used with a non-unique index on *ename*, the following SELECT statement results:

```
SELECT empnum, ename, rowid
FROM emp
ORDER BY ename ASC,ROWID ASC
```

Because the ROWID or Position Index was added to the ORDER BY clause there is no index in the database that satisfies this order, and a sort will be performed in the database. Therefore, it is best to use unique indexes.

**Note:** Magic tries whenever possible to avoid adding the ROWID or Position Index columns to the ORDER BY clause.

For example, in Oracle batch programs that use a non-unique index, the rows are scanned in only one direction. Therefore, Magic does not add the ROWID to the ORDER BY clause. Therefore, the Position Index is added to the ORDER BY clause when a non-unique index is used.

## ***A Simple Magic Program***

The table below contains data for an example program.

**Table name: emp**

<b>empno</b>	<b>ename</b>	<b>Rowid</b>
1	John	11111
2	Benjamin	22222
3	Bill	33333
4	Tom	44444

The first unique index for this table is empno. When you run the program Magic does the following:

- Step 1 - Opens a cursor for

```
SELECT empno,ename,rowid
FROM emp
ORDER BY empno ASC
```
- Step 2 - Creates a loop for retrieving all four rows to be retrieved to the screen, until the end of the rows

- Step 3 - Performs an internal Magic function called GET CURRENT' to retrieve the first row and position on it using

```
SELECT empno,ename,rowid FROM emp
WHERE rowid=11111
```

Every time the user scrolls on the rows on the screen, Magic performs the GET CURRENT operation for the rows being scrolled.

If the rows would have been cached, the GET CURRENT operation would have been performed from the cache and not from the database.

If the user wants to update a row, depending on the locking strategy, Magic performs an internal function called *hook* that retrieves the row again from the database by issuing:

```
SELECT empno,ename,rowid FROM emp
WHERE
rowid=11111 FOR UPDATE
```

In Oracle, Informix, and DB2 the hook is performed with a FOR UPDATE clause.

In Sybase, MS-SQL, and ODBC the hook is performed without the FOR UPDATE clause.

## ***Regular Link Operation***

For each row being read from the main table, Magic adds a second SELECT statement to retrieve the linked row.

The rows for the main table are read by a loop that fetches all the rows as described above. To perform the link, every time the FETCH operation of a row in the main table is performed, another cursor is opened to retrieve the rows of the linked table.

If there are two links on a main table, Magic opens a cursor for the main table and in a loop:

- Fetches the main row

- Opens a cursor for the first linked table
- Opens a cursor for a second linked table

The loop ends at the end of a table or at the end of a screen.

A Link operation is different than a Join operation:

- if the row in the table does not exist in the joined table, the row will not be retrieved.
- A Join of several tables makes them into one entity, and only one cursor will be opened and retrieved.
- A link in Magic results in slower response time than a Join because the link communicates and requests more from the SQL server than a Join.

It is advisable to:

- Use a view that joins tables in reports and read-only queries.

Cache the linked tables, especially if the linked tables contain only a few rows, or if the linked tables result in repeatable identical rows.

## ***Link Join Operation***

The purpose of the Link Join operation, is to establish a many-to-one dataview relationship between the main table and the linked joined tables, that will be implemented by the underlying RDBMS. The operation will generate an SQL inner join statement among all the participating tables.

### ***The Join Operation***

The Join Operation, is a robust approach to the use of RDBMS Join capabilities, for the reading and locking data in conjunction with Magic mechanisms. When the user updates a row or when Magic recompute is taking place, the Link Join operation behaves as a Link Validate operation.

## ***Join Operation Parameters***

The Join Operation parameters are displayed below:

---

<b>Parameter</b>	<b>Meaning</b>
Table Identification Number	Zooming from this column will display a selection list of all tables from the same database as the main table.
Index	This index should be used with this table. Since only many-to-one relationships exist between the main and the linked table, only unique indexes can be selected. Zooming on this column will bring a selection list of all the unique indexes that belong to the linked table.
Direction	Not used
Return	Enables you to specify a variable that receives the return code. This indicates whether or not the link succeeded. When the Link Join generates a Join Statement, it will always return 'True'. When the Link Join behaves as Link Validate (due to recompute or update), the return code is the same as defined for Link Validate.
Condition	Same as in Link Validate

---

## ***Join Operation Usage***

1. All the index segments of the joined table must be selected and each index segment should have a Locate expression.
2. An exact range is needed for the joined tables. Therefore, the Minimum Locate expression must be equal to the Maximum Locate expression.
3. The following locate expressions on Joined tables are presented below:
  - A constant.
  - A variable reference to a column from the main table that was selected in the task.
  - A variable reference to a column from a joined table that was previously selected in the task.
  - A non-complex expression (no complex expressions allowed).

## ***Join Operation Behavior***

When a dataview is created for the task, Magic generates a joined statement for all of the joined tables. Magic creates a WHERE clause which consists of the constants and the matching clauses, with the names of the columns from the main table and the joined table. Then Magic builds the rest of the dataview.

### ***Locking***

When Magic needs to lock a row, it generates a lock in the underlying RDBMS.

In SQL gateways that use logical locking, the logical locking is done simultaneously for the main and joined tables.

In SQL gateways that use physical locking, the SQL gateway checks whether the RDBMS allows a FOR UPDATE clause for a joined table SELECT:

- If the RDBMS such as Oracle, allows a FOR UPDATE clause for a joined table SELECT, then the lock is performed simultaneously for the main and joined tables, using the FOR UPDATE clause. This is done to lock only the joined tables which have WRITE access mode in the task.
- If the RDBMS such as DB2 or Informix, does not allow a FOR UPDATE clause for a joined table SELECT, then the lock is performed by different SELECT FOR UPDATE statements, for each table which participates in the Link Join.

If a join table is opened in the task with read access, Magic will try to lock only the main table and not the join table.

When Magic needs to update the database, it updates each joined table separately according to the position index for that specific table.

### ***Oracle Locking Example***

Magic checks the Access/Share mode of each table participating in the Join. For each table requiring a lock, Magic adds one of its columns with the OF clause of the SELECT FOR UPDATE statement.

For example, 'SelectA.col1, A.rowid, B.col1, b.rowid from Table1 A, Table2 B where A.col1=B.Col1 and A.rowid='current row number' FOR UPDATE OF a.rowid'.

This example specifies that only the row from Table 1 will be locked.

### ***DB2 and Informix Locking Example***

When physical locks are used, Magic tries to issue: SELECT...FOR UPDATE statement. However, in DB2 and Informix the FOR UPDATE clause is forbidden when the ORDER BY clause is used and/or Join tables are used. Therefore, when using Batch Task with Immediate Locking Strategy for a non query task, Magic opens a cursor for the Joined tables.

For example, Select A.co11, A.rowid, B.co11, b.rowid from Table 1 A, Table2 B where A.col2=B.Col2 ORDER BY A.co11, and then for each row in each table, the result is a separate SELECT...FOR UPDATE used.

Therefore, it is recommended to use a different locking strategy in order to enhance performance.

### ***Usage Considerations***

1. The Link Join operation is valid only for SQL databases. Only tables from the same database can be joined.
2. Link Join differs from Link Validate in the following ways:
  - In Link Validate, a sub-select is done to fetch each linked row. If a linked row is not found, the corresponding row from the main table is still available.
  - In Link Join, an inner join SELECT statement is used for the main and joined table. For this reason, integrity constraints are important.
3. Because DB2 and Informix require separate locks for each table, Link Join in batch tasks with immediate locking strategies, are implemented as Link Join without a lock, and as Link Query/Validate, for each row which is fetched.
4. When Magic generates a SELECT statement with multiple tables, Magic uses aliases for the table names, such as in A for a main table, B for the first join, etc.
5. Cache on Join Task

The Cache strategy is based on the setting of the Cache strategy property on the Advanced tab in the Task Control dialog. In this special case, the cache rows are located on the view that contains the main table and all link join tables. This is the reason that there is no meaning to the Cache property of the linked table itself, in the DB Tables Form. Due to this special behavior, the cache is handled differently.

### **Insert**

When you create a new row, this row will be inserted to the database, but not to the cache. It will be inserted only after a Fetch or Get Current of this row will be issued from the database.



## **Delete**

When you delete a row, it will be deleted from the database and then from the cache as well.

## **Update**

When you modify columns in a row of your view, one or more update statements can be sent to the database for each table that has been updated.

When the update is on the main table, the row will be deleted from cache. It will be inserted again only after a Fetch or Get Current will be issued for this row. This is because the row's update might change the column that links the main table to the link table - and while doing the update itself, the link row values are unknown to the cache.

When the update is on the linked table, the cache will be erased and every row that is fetched will be inserted to the cache. The reason is that this update might affect other rows in the link join, if there is more than one record of the main table which is linked to the same linked table row, and these rows cannot be located by Magic according to the linked table position.

The cache is also released in the following cases:

1. Changing the task mode to create.
2. Performing a user sort option.
3. Performing a user locate option.
4. Performing a user range option.
5. Changing the index.

## Ranging

When a Range is being issued by the user or programmer, Magic adds the correlating WHERE clause to the statement.

It is important to match the Range to the Index, so that the optimizer will not perform a sort in the database.

### **CNDRANGE( ) Function**

The new CNDRANGE function allows a conditional Range in the minimum and maximum Range, and locates expressions.

When the condition parameter is evaluated as False, the Range or Locate clause is not generated. This function can be used when minimum and maximum values in the expressions do not retrieve columns with null values, or cause a penalty in performance.

**Syntax:** CNDRANGE (*condition*, *value*)

**Parameters:** *condition*: Boolean expression which is evaluated during runtime

*value*: Value to return if the condition is true

**Returns:** Any value as specified in the parameter list, if the evaluation of the <M> *condition* parameter was true. Note that if the condition parameter evaluates as false, there is no return value or action.

#### **Example:**

CNDRANGE ('TRUE'L, 10)

The function will return the value 10.

CNDRANGE ('FALSE'L, 20)

The function will remain idle (as if no expression was given).

**Note:** The function can only be used for locate and range expressions in a select operation. The attribute of the value must be compatible with the attribute of the selected column.

## **SQL Where Clause**

The purpose of the SQL Where feature is to enable the programmer to use the SQL-specific Where clauses (in addition to the Where clauses generated automatically by Magic) without the need to use the Direct SQL tasks, and to view the Full Where clause that is generated by Magic.

This is an interface for writing Range based on SQL syntax using Magic columns. This range is added to the Magic Where clause that is defined in the Record Main.

### **SQL Where Usage**

The SQL Where clause can be specified by using a form. This form consists of the following components:

- Column list
- Add to Where clause area
- Full Where clause view

#### **Column List**

The variable list is a list of all the variables available to use in the SQL Where clause:

- Real columns from the main and joined tables - will be replaced with their DB column name.
- Variables from parent tasks - will be replaced with their values.
- Virtual and other real columns from the current task - will be replaced with their values.

#### **Add to Where Clause**

The Where clause refers to a free-text form in which the SQL Where is written. Three types of strings can be written in the Where clause area. They are as follows:

- A column number (A,B,C, etc.) prefix with a : sign.  
If the column is from either the main or joined tables it will be replaced by its DB column name. Otherwise, it will be replaced by its value according to its attribute. For alpha columns, Magic will suppress any trailing blanks, and add quotes to it.
- A column number (A,B,C, etc.) prefix by @: sign and is an **Alpha column not** from the main or joined tables. It will be replaced with its value without adding quotes to it.

### Full Where Clause

Magic displays the entire WHERE clause as used in the SQL statement generated by Magic:

- The Where clause generated from the Record Main, including all the 'From' and 'To' expressions.
- The SQL Where clause, added in parenthesis with an AND clause, to concatenate it to the Record Main Range.

The full WHERE clause replaces every occurrence of a column from the column list. Real columns from the Main and Joined tables, are replaced in the format of 'column DB name' or 'A.column DB name, and virtual fields are replaced with their description.

When Magic replaces a column with its contents, Magic checks the column's attribute and storage, and if necessary adds quotes - as in the case of alpha strings. In order to prevent Magic from adding quotes to the alpha columns, enabling any type of syntax to be written, the @ character should be added as a prefix to the column.

The SHOW button at the bottom of the screen refreshes the Full WHERE clause, including the join conditions from the Link Join.

For example, assuming

A is real column with DB name Employee.jobname, and

B is a virtual alpha column with description Vjobname, and its value in Runtime is "AB"

then writing the SQL range

:A like 'B%' will be displayed as

Employee.jobname like 'B%'  
(the table name will be added only when  
link join is involved)  
and will translate in Runtime to  
jobname like 'B%'

Writing the SQL range  
:A like :B will be displayed as  
Employee.jobname like ["Vjobname"]  
(the table name will be added only when  
link join is involved)  
and will translate in Runtime to  
jobname like 'AB'

Assuming that C is a virtual column, and  
its Description (Name) is Voperation  
and its value in Runtime is "like"  
then writing the SQL range  
:A @:C :B will be replaced with  
jobname [Voperation] ["Vjobname"]  
and will translate in Runtime to  
jobname like 'AB'

## ***SQL Where Behavior***

When a dataview is created for a task, Magic generates a Select statement with a Where clause. The first part of the WHERE clause is based on the From and To Where clause expressions, in the Record Main. The second part of the WHERE clause is the SQL Where, which is added to every SELECT statement. The column's values are evaluated only **once** when entering the task, and no recomputing is done. The only time that the SQL Where is not used, is when Magic generates the SELECT statements for the internal *Get Current* and *Hook* (determined by the locking strategy) operations. In these operations, Magic uses the position index to get the row, and the SQL where is not needed.

## ***SQL Where Usage Considerations***

- When using virtual columns from the current task in the SQL range, the virtual columns must receive their values from the calling task's parameters, otherwise their default values will be used.  
**Note:** The *init* expression is calculated afterwards, therefore it cannot be used.
- Blobs and Memo fields cannot be used in the SQL range.
- Magic does not check the syntax of the SQL range. Therefore, if an invalid SQL syntax is used, a message from the RDBMS will occur in Runtime.
- Magic does not check the attribute of each column used in the SQL Where string. It tries to convert each attribute to a string which is concatenated in the corresponding location in the SQL Where string. Therefore, it is important to use the attributes properly.
- When the Link Join operation is used, Magic uses an alias for the Table names. Therefore, instead of Table1.Column1, it uses A.Column1. If a string is written in the SQL which contains a column name, the alias letter corresponding to the Table should prefix the column name.
- Magic suppresses any Trailing blanks. For example, if a virtual column, Vco11 is Alpha 10, and its value is 'ABC', then writing **:Vco11** will translate in Runtime to 'ABC' and not 'ABC '.

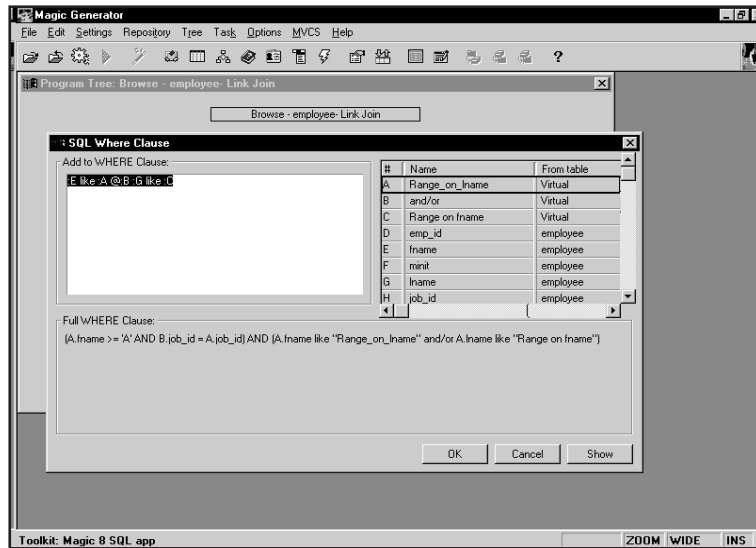


Figure 19-1 SQL Where Clause

## One-Way and Two-Way Behavior

One-way and two-way index behavior is relevant only when locating in online tasks.

As described previously, RDBMSs, except for Rdb V6, support only one-way indexes. When we want to locate on a specific row, Magic has to perform three steps:

- **Step 1** - Look for that value using:

```
SELECT empno,rowid
FROM emp
WHERE empno=3
ORDER BY empno ASC
```

- **Step 2** - Display all the rows below the specific row on the screen using:

```
SELECT empno,rowid
FROM emp
WHERE empno3>=3
ORDER BY empno ASC
```

- **Step 3** - Display all the rows above the specific row on the screen using:

```
SELECT empno,rowid
FROM emp
WHERE empno<=3
ORDER BY empno DESC
```

When there is a one-way index:

- The row displays on the top of the screen.
- Only Step 1 and Step 2 are performed.
- A buffer of up to 500 row IDs is kept for scrolling up.
- You cannot scroll above the desired row.

When there is a two-way index:

- The behavior depends on the flag.
- If Locate has been defined in the MAGIC.INI file to be in the center of the screen, the row displays in the center of the screen, and all three steps are performed.
- The descending order required by step 3 might cause a database sort, which lowers performance.

This is due to the fact that for some RDBMSs, Indexes cannot be traversed backwards.

When using a one-way index, the Ctrl +End, used in the two-way index to bring you to the end of the table, will not perform any operation.

### **Recommendations:**

- Use a one-way index.



- In Informix, if you must have a two-way index, create a descending index. This can only be done in Informix.
- Beginning with Magic 7, if you use a two-way index, set the Center Screen In Online flag in the MAGIC.INI file to No, so that step 3 will be performed only if the user explicitly tries to scroll up.

## ***Incremental Locate***

When you use Incremental Locate, every time you type a character Magic re-issues all the SELECT statements described in the steps above.

It is advisable to avoid the use of the Incremental Locate operation in your applications.

## ***Sort Using RDBMS***

When using SQL databases, the RDBMS engine can sort the rows dynamically by using the ORDER BY clause, without the need to define a index or index as is required in ISAM files.

Sort usage in Magic remains unchanged. However, when all columns are sorted as part of the Main and Joined tables, Magic allows the RDBMS to sort. When it is not possible for the RDBMS to sort, Magic sorts the rows using a temporary file as done in earlier versions of Magic. Sort usage in a task is the same as defining a virtual index for that table. In addition, it will save the programmer from having to define many virtual indexes for any combination that the user might want.

## ***Sort Usage***

As in the prior versions of Magic, the user defines the sorts segments which will be used.

## ***Sort Behavior***

When defining a sort, in which all segments are part of the Main and Join tables, Magic will reissue the SELECT statement for that task, and will replace the ORDER BY clause that was generated according to the Index, with the column names used in the Sort.

Magic always requires a unique ORDER BY clause. If the user did not specify that this ORDER BY is unique, and if Magic did not determine that the sort segments in combination were unique, Magic will add the position index segments to the ORDER BY clause. In addition, if any part of the columns selected in the sort were not part of the Main and Join tables, Magic will create a temporary sort file.

## ***Sort Usage Considerations***

When you define a sort that results in an ORDER BY clause, the RDBMS might not use an Index to perform a Query.

# *Debugging Your Programs*

# 20

---

**I**t is very easy to track the exact statements that Magic passes to the SQL database because all Magic interactions with the RDBMS are performed in the SQL language. It is also easier to tune the Magic application or the database parameters to achieve maximum performance in SQL than it is with ISAM files.

## ***Magic Gateway Log***

All the Magic SQL gateways were written using a template that enables you to monitor all the different SQL gateways in the same way.

### ***How to Set the Flags***

In earlier versions of Magic, an External SET command was needed to set the Log File on. In Magic 8, it will be necessary to simply turn the log on in the DBMS properties dialog.

In VMS and OpenVMS use the DEFINE command.

In Windows, Windows 95, and Windows NT use the Magic DBMS properties to set these flags.

## The Flags

The log file Name can be any valid Name. The Log Level can be one of three levels of information which are written to the log file:

- **Customer Level**—contains information which is relevant to the flow of the programs, such as: the environment, versions, the connect string at the beginning of the log file; the SQL statements which are generated by the gateway, and the beginning and commit transactions. It is recommended to use level 3 to track the execution of your programs.
- **Developer Level**—contains information which includes: a timestamp, values, and entry exit points to the Magic File Manager routines.
- **Support Level**—contains detailed information that the MSE technical support team uses to locate problems.

### Log Sync. (Yes/No)

This flag should remain at its default setting of No. If there is a serious problem that causes the gateway to crash, the value should be set to No, in order to find the exact location in the application that caused the gateway to crash.

**Note:** setting this flag to No slows down performance.

### Show plan (Yes/No)

This flag is relevant only to Sybase and MS-SQL. Setting this flag to Yes causes the gateway to enable the show plan option. The access path to the data is then written to the log file, enabling the user to see whether indexes or sorts were used. This is a good tool for tuning your application without using RDBMS tools.

In DB2, setting this flag to Yes will cause DB2 to write the Trace to the DB2 Trace table, in which the access path can be viewed using DB2 tools.

## ***Where to Set the Flags***

The flags should be set in the part of Magic that contains the full SQL gateway rather than the definition gateway. In the Magic Client/Server environment these flags are set on the server side and not on the client. If you are using the Open Client/Server architecture, the setting is on the client side.

## ***How to Work with the Log File***

Set the flags and start your application. After every step, switch to the log file using any browser, without quitting Magic. A new log file will be started every time you start Magic, and the log file will be closed when you exit Magic. Now you can perform a few steps in Magic at any given point in time and switch to the log file from your last location.

If the log gets too big, you can exit Magic and then enter Magic again so that the log will be recreated.

## ***RDBMS Tools***

As explained earlier in The Optimizer section, there are tools for each SQL server to track and analyze the access path of the optimizer for a specific SQL statement or a set of SQL statements. You can apply these tools to your application by setting the right flag or setting that lets you log your SQL commands.

- Informix - Run a direct SQL program with the command:

```
Set explain on
```

When this program is executed, all the information is logged to the log file sqexplain.out.

- Oracle - Run a direct SQL program with the command:

```
alter session set SQL_TRACE=true
```

When this program is executed, all the information will be logged, and the TKPROF utility will produce a readable trace file.

- MS-SQL and Sybase - The use of the SQL\_SHOWPLAN in Magic does the work without any need for an external tool. In MS-SQL 6.5 there is an SQL Trace program to externally trace the command.

Other monitors on the amount of locks, loads, memory utilization, and usability of the SQL servers are available, and should be used to verify that your SQL server is properly configured.

## ***Magic Profiler***

The Magic Profiler is a powerful tool that lets you debug your programs in Runtime. It should be used in the Magic SQL environment as well to isolate problems relating to task performance timing, and to clarify whether the problem is in the application or in the SQL interface.

## ***Flow Monitor***

The SQL Gateway logs can also be viewed from the Flow Monitor.

# *Locking and Transaction Processing*

---

# 21

**L**ocking and transaction processing are essential in multi-user environments. To achieve maximum concurrency, online transactions must be short and must not interfere with user interaction.

Transactions are opened according to the Magic task's parameter settings, which can be set at the program or the system level. The parameters that affect the way Magic opens transactions and enforces locking include:

- Environment settings for multi-user and ISAM Transaction
- Database settings for table locking
- Access and share mode in the database table repository
- Locking strategy options specified in the Task Control dialog
- Task type specified in the Task Properties dialog: Online or Batch
- Transaction-level settings
- The On Record Locked and transaction Error parameters
- Call Task or Program LCK parameter
- In the system's Environment settings, both the Multi-user Access parameter and the ISAM Transaction parameter must be set to Yes. Otherwise Magic will not issue any transaction that handles requests.

## ***The Multi-user Flag***

In a Runtime environment, the Multi-user flag specifies that when a Magic lock is performed, the lock is also performed inside the underlying database.

When the Multi-user flag is set to Yes, Magic also sends the necessary SET TRANSACTION commands to the DBMS to ensure data integrity with other running applications.

When the Multi-user flag is set to No, the application cannot be used simultaneously by more than one user. Therefore, no locks are performed by Magic, and no physical or logical locking is performed by the gateway (that is, no SELECT ... FOR UPDATE).

When the Magic Locking parameter is set to No, Magic will not use its own locking mechanism to implement file level share mode, explained in the next section.

## ***Table Access and Share Mode***

In Magic you open a table with both an access mode and a share mode. When opening a table, Magic performs some internal tasks, such as getting the structure of the table from the database, checking for the table's existence, and more. If Magic locks are used, relevant information is written in the mglock file.

When working with RDBMSs, there is no Open File or Open Table command. Therefore, the access and share modes have almost no meaning except for the following cases:

- Share is None - The SQL gateway sends a Lock Table command to the SQL database. This is done only in RDBMSs that support the SQL statement "lock table ..."
- If the access and share modes are READ and WRITE, and the table is a linked table, when a lock is requested the records of the linked tables will not be locked. If the access is WRITE Magic assumes that the linked table may also be updated, and a lock is issued for it as well.



# Magic Locking Strategies

Magic locking strategies are set per task in the Task Control dialog, as shown in Figure 21-.

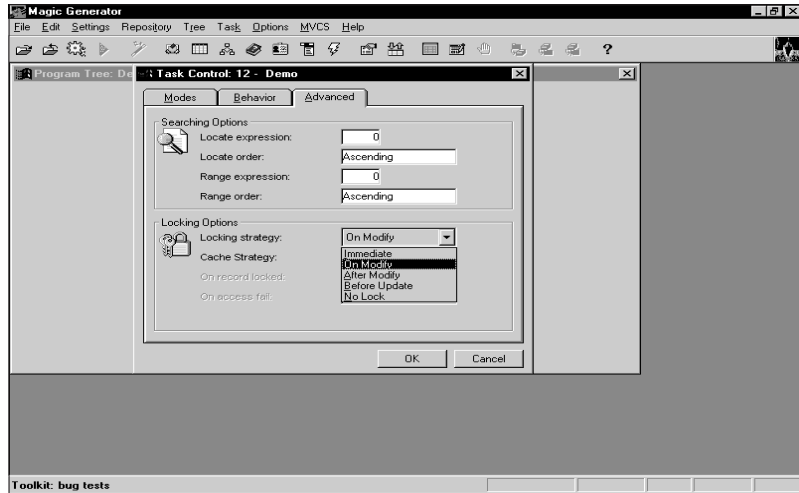


Figure 21-1 Locking Strategy Parameter - Task Control

In all locking strategies except Immediate, Magic reads the record for display without locking it the first time. At a later stage, the record is reread using a lock on the record. The process of rereading the record is called *hook-up*.

During hook-up, Magic rereads the record using the record's position and verifies that all the selected values are the same as when the first read was performed. If the values are not the same, an error message is issued. The changes introduced by the process are then rolled back, and the user needs to restart the update.

## Immediate

Magic locks the record as it is read, which is the default option for batch tasks. Locking the record as it is read can cause prolonged record locks in

online tasks in multi-user applications because one user may remain in a program for a considerable amount of time.

In batch tasks, Magic opens the cursor with the FOR UPDATE clause, which may harm performance in Oracle and Informix.

- In Oracle, opening such a cursor may take time.
- In Informix, the FOR UPDATE clause is not allowed with the ORDER BY clause. Therefore, every record fetch starts by opening a cursor on every record with a FOR UPDATE clause.

## ***On Modify***

Magic hooks up to the record as soon as the user begins updating the record.

If the On Modify locking strategy is selected, in online tasks locking occurs when the first character is typed into an input field or selected from a lookup table. On Modify is the default option for online tasks.

In batch tasks, locking occurs with the first Update Var operation executed in record suffix, or the first task or program called.

## ***After Modify***

In using the After Modify locking strategy, a whole field must be entered before locking occurs. Magic locks the row only before the actual write to the database. This means that the row is not locked during the interaction stage (Online) nor during the Record Suffix operations (Online & Batch). In After Modify, the user has to fill a complete column, and move to a different column before the lock is tried. After Modify should be used with care, in order to prevent data integrity problems. Use After Modify where the locking operation of the underlying database carries a heavy toll, and where the task structure ensures that no data will be left inconsistent, should the verification fail. This option exists for compatibility with version 4.xx and version 5.0x.

## ***Before Update***

When the Before Update locking strategy is selected, Magic hooks up to the record before the actual write to the database. Subtasks called by the current task may have updated information to other tables, potentially causing inconsistency if the current record is not updated to disk when this option is selected.

The Before Update option should be used with caution to prevent data integrity problems. It is best to use this option only where the locking operation of the underlying database carries a heavy toll, and where the task structure ensures that no data will be corrupted if verification fails.

## ***No Lock***

The No Lock strategy is similar to the Before Update strategy in lock timing, but the hook-up is not performed. The record in memory is updated to disk, even if the record has been updated by a different user in the time period between the initial read and the write back to disk operation. In this event, the write operation overwrites any other user modifications without checking them.

The No Lock strategy should only be used when it is clear that no other user can access the record while the record is being processed by the current station. For example, if an application level lock flag, which performs locking similarly to the way locking was performed in Magic before the databases handled locking, signifies that an object is being updated, then the No Lock strategy is sufficient.

In versions prior to Magic Version 7, the No Lock strategy is called the Minimum Locking strategy. In these versions, the record is read again before writing it, but the record is not compared to the earlier values. The No Lock strategy corrects this behavior.

It is best to use the No Lock strategy unless the time duration between the record prefix and the end of the record suffix is long enough to allow other users to change the record. In this case, use the Immediate locking strategy.

# Transaction Levels

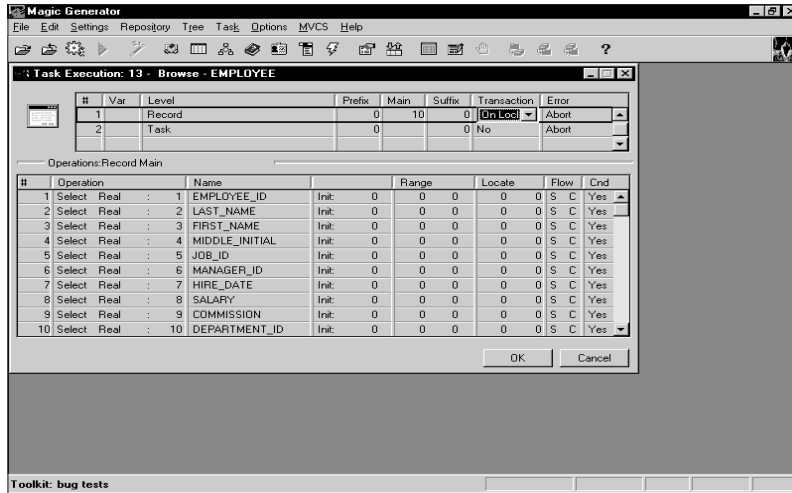


Figure 21-2 Task Execution Table

## Task Level

Transactions can be specified at the task level as Yes or No. When set to Yes, the transaction is set at the task level. If all the records processed in a task are to be processed together, use a task level transaction.

**Please note, Magic does not support nested transactions.** If a task level transaction is declared for a subtask or subprogram that is activated within a previous transaction, the gateway will ignore the second transaction.

## Record Level

The Transaction parameter in the Record Level row can have five different values: On Lock, Prefix, Suffix, Update, and No.

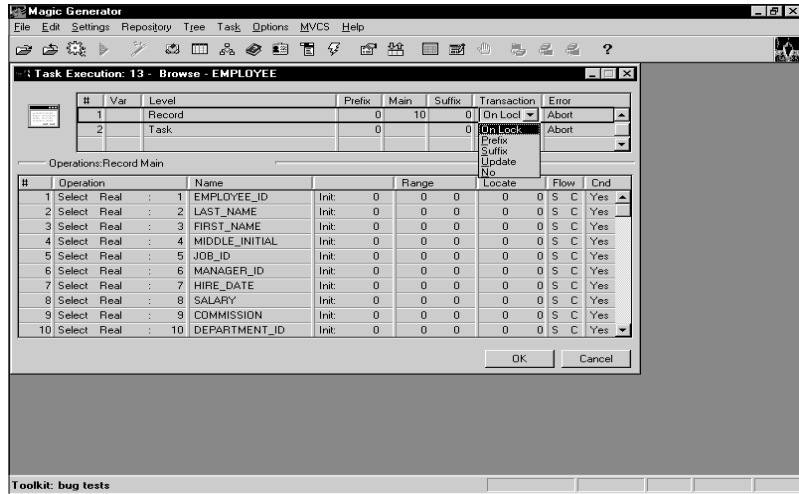


Figure 21-3 Record Level Transaction Modes

## ON Lock

This is the Default Transaction mode, specifically the Transaction begins as the Lock is issued.

## Prefix

When you define the Transaction parameter as Prefix, Magic:

- Opens the transaction before fetching the record
- Commits the transaction only after the data table is updated, which occurs after the Record Suffix is executed
- Includes all operations between the open operation and the commit operation, including subtasks, within the transaction scope

When the transaction is defined at Prefix level, the transaction is open during the interactive stage (Record Main). You should therefore not define a transaction at Prefix level for online tasks in a multi-user environment,

because the transaction involves locking and may cause problems for other users trying to access the same records or table.

### ***Suffix***

When you define the Transaction parameter as Suffix, Magic:

- Opens the transaction before executing the Record Suffix
- Commits the transaction only after the dataview is updated, which occurs after the Record Suffix is executed

All the Record Suffix operations, including subtask calls, are included in the Suffix level transaction.

In case of transaction failure, the engine retries execution.

### ***Update***

Update is the default Record level transaction for online tasks. When you define the Transaction parameter as Update, Magic:

- Opens the transaction after the Record Suffix is executed, just before the data table is updated
- Commits the transaction only after the data table has been updated

A Record Level transaction defined as Update does not include Record Suffix operations. The Update transaction is best for multi-user applications because it lowers the risk of concurrency problems among users.

### ***No***

Defining the Record Level transaction as No specifies that no transaction is required at Record level.

## Transaction Level Rules

The way that Magic versions 5, 6, and 7 open a transaction can be summed up in the following statement:

When a lock is requested, Magic opens a transaction when that lock is issued. In all other cases, Magic opens a transaction according to the transaction definition.

However, in Magic 8, if a lock is requested before a transaction is opened, as in lock immediate suffix transaction, the following error will be displayed:

“Dataview cannot be locked outside the scope of a transaction.”

## Error Recovery Control Options

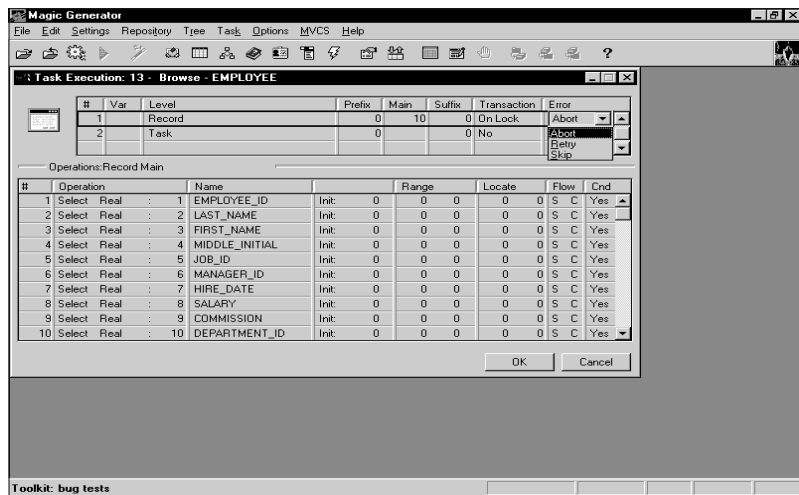


Figure 21-4 Error Recovery Control Options

Error recovery control is determined by the On Record Locked parameter or by the transaction Error parameter.

## ***On Record Locked Parameter***

The lock transaction is issued for locking purposes only. If an error occurs during batch processing after a lock transaction has been issued and before the designed timing of a Read/Write transaction has ended, and if the error is related to waiting for a locked record, then error recovery control is determined by the batch task's On Record Locked parameter as defined in the Task Control dialog.

The On Record Locked parameter defines Magic's behavior when it encounters a locked record during batch processing. The available options are: Abort, Retry, and Skip.

### ***Abort***

Magic aborts the task.

### ***Retry***

Magic retries the lock until the record is released by the user who has locked it. This allows processing to continue after the problem is resolved. During the retries, the user may be able to abort the task, if allowed. Retry is the default setting.

### ***Skip***

Magic skips locked records, and continues processing the next available record. When using the Skip option, processed and unprocessed records should be marked in some way so they can be identified in a second pass. Remember, the locking strategy is actually implemented in the best way the RDBMS supports.

## ***Transaction Error Parameter***

If the error occurs after the designed timing of a Read/Write Transaction has ended, error recovery control is determined by the transaction Error parameter.



If the error is not related to locking, the error is handled using the task's On Access Fail parameter as defined in the Task Control dialog.

## ***LCK Parameter - Call Task or Call Program***

As show in Figure 21-, when a task or program is called, Magic locks the current record by default, even if a lighter locking strategy has been selected.

## ***Physical and Logical Locks***

There are two major levels of locks in SQL:

- An exclusive lock, which is automatically generated when an UPDATE statement is issued.
- A shared lock, which can be issued by a SELECT statement or by adding the FOR UPDATE clause at the end of a SELECT statement. A shared lock tells the SQL server that you plan to update the record, and that no updates will be allowed until this lock is released. If another user tries to update the record or to send a SELECT ... FOR UPDATE statement, an error message that the record has been locked by another user is sent to that user.

### ***Physical Locks***

The SELECT ...FOR UPDATE statement is available in Oracle, Informix, and DB2. These RDBMSs also support row-level locking. Therefore, when a lock is requested according to the selected locking strategy, Magic tells the gateway that the record should be read again with a lock. If a transaction has not been started, the gateway starts a transaction. Then the gateway reads the current record with the FOR UPDATE clause.

The procedure described above ensures that no application, including non-Magic applications, will be able to update the record until the end of the transaction, which usually occurs after the update is done.

### Example:

The current record is retrieved:

```
SELECT empnum,ename,deptno,rowid
FROM emp
WHERE
    rowid=1111
    values: 1      ,      John,      30
```

Lock is requested:

```
SELECT empnum,ename,deptno,rowid
FROM emp
WHERE
    empnum=1111
    FOR UPDATE NO WAIT
    values:      1      ,      John,      30
```

Assume that the deptno was changed to 40:

```
UPDATE emp SET deptno=40
WHERE
    rowid=1111
```

### ***Batch Immediate Locking Strategy***

In batch tasks, when an Immediate locking strategy is used the gateway may be able to lock the whole dataview.

In Oracle, a SELECT statement can be issued with a FOR UPDATE clause and an ORDER BY clause. Therefore, when the cursor is defined at the beginning of the task, it is declared with a FOR UPDATE clause. Then all the records are fetched from this cursor.

Informix and DB2 do not allow the procedure used by Oracle. Instead, the cursor is opened, and a cursor is opened with a FOR UPDATE clause for every record.

Please refer to the Link Join section for additional information.

In Magic versions prior to Magic Version 7, the Immediate locking strategy does not cause a FOR UPDATE clause.

## Logical Locks

Sybase and MS-SQL do not have the FOR UPDATE clause and support only page level locking, which may result in locking problems. Therefore a logical lock strategy is used in which the record is not actually locked. Instead Magic verifies, for integrity reasons, that no one changes the record from the minute the record was logically locked until the update.

When a lock is requested and Magic asks the gateway to read the record with a lock, the gateway reads the record and keeps the values of the read record. When the UPDATE statement is then issued in the record suffix, all the columns, called fields in Magic, are added to the WHERE clause.

If in that period of time the record, which was not locked, has been changed by another user, the gateway sends a message that the record has been changed by another user, and the UPDATE fails.

### Example:

The current record is retrieved:

```
SELECT empnum,ename,deptno
FROM emp
WHERE empnum=1
values:      1      ,      John,      30
```

Lock is requested:

```
SELECT empnum,ename,deptno
FROM emp
WHERE empnum=1
values:      1      ,      John,      30
```

Assume that the deptno was changed to 40:

```
UPDATE emp SET deptno=40
WHERE
empnum=1 AND ename='John' AND deptno=30
```

The ODBC gateway also uses logical locking behavior because it cannot assume that a record lock or a FOR UPDATE statement is available in the accessed database.

## ***BLOBs Locking***

Magic does not lock a record in the database when only the BLOB field was modified because in most RDBMSs you cannot add the BLOB field to the WHERE clause. You can automatically update another field every time the BLOB field is updated to enforce database locking.

## ***Isolation Levels and Optimizer Hints***

A flag that lets you determine the isolation level to be used in the application has exists for Sybase and MS-SQL, to enforce or avoid locking.

You may set the isolation level to allow DIRTY READS for maximum concurrency or use level 1 for READ COMMITTED.

You may also use the HINT flag to hint to the optimizer about what kind of lock to issue, and not only which index to use.

The default options of these flags are recommended in most cases. The defaults should only be changed in special cases by someone with a good understanding of the underlying RDBMS's locking mechanisms.

## ***The ROLLBACK Function***

When working with transactions, the application's logic may need to roll back transactions.

A transaction is defined to ensure integrity and to ensure that several actions will be treated as one.

Magic supplies a function called ROLLBACK that has two parameters:

- A logical variable determines whether to display a confirmation window in Runtime - True or False.
- Level of transaction should always be 0 because Magic supports only one level of transaction.

For example: ROLLBACK ('True'Logical,0)

The ROLLBACK function may be used in any task. It will roll back everything to the beginning of the transaction, even if the transaction was started high up on the task tree.

The ROLLBACK function is a very powerful tool. If a rule of the application logic has been violated, or a crash or power failure occurs, the user can be returned to the starting position.

The ROLLBACK function should be used with caution. For example, if the transaction is long a lot of user work will be rolled back, so it may not be desirable to use the ROLLBACK function.

If the ROLLBACK function is called and there is no outstanding transaction, nothing happens. If a confirmation window is used, the confirmation window will not appear.

The ROLLBACK function can be used to create an application event that activates a ROLLBACK program. The user can then use a hot key or pulldown menu to roll back a transaction at any time.

## ***The Intrans Function***

The INTRANS function evaluates if a transaction is currently opened. The function has no parameters, and returns a Boolean True-False value indicating if a transaction is in progress or not.

Syntax: INTRANS ()

Parameters: None

Returns: Boolean

[This page intentionally left blank]

**W**hen SELECT statements are complicated, it is faster to let the RDBMS server join and constrain the rows, bringing only the specified rows into the Magic task's dataview. This is especially helpful in a client/server environment, where lowering network traffic improves overall system performance.

An RDBMS can perform vertical updates and deletes with one SQL statement, one cursor, and a simple transaction. Magic tasks process the dataview one record at a time. In an Insert, Update, or Delete task, each record is processed against the server separately, even though all the records may be processed as one transaction.

You can use explicit SQL where DDL operations are runtime-specific. For example, you may need to create a special table index for a specific report and then drop the index when the report is complete. Or, you might want to create a temporary table in the RDBMS .

In general, performing other types of DDL from Magic is not recommended. RDBMS joins are usually more efficient, although not in all cases.

# Using Explicit SQL

Magic lets you embed native SQL statements in your Magic application with the Task Object SQL Command. If the SQL Command object is used, the task's Main Table is the SQL statement.

## Open the Task menu

1. From the Tools Menu, choose Programs.
2. Create a new line.
3. Type demo for the program name.
4. Zoom to the task tree.
5. Zoom to the Task Properties dialog.
6. From the Task Prefix field, click the Task menu.

The SQL Command dialog, shown in Figure 22-, appears.

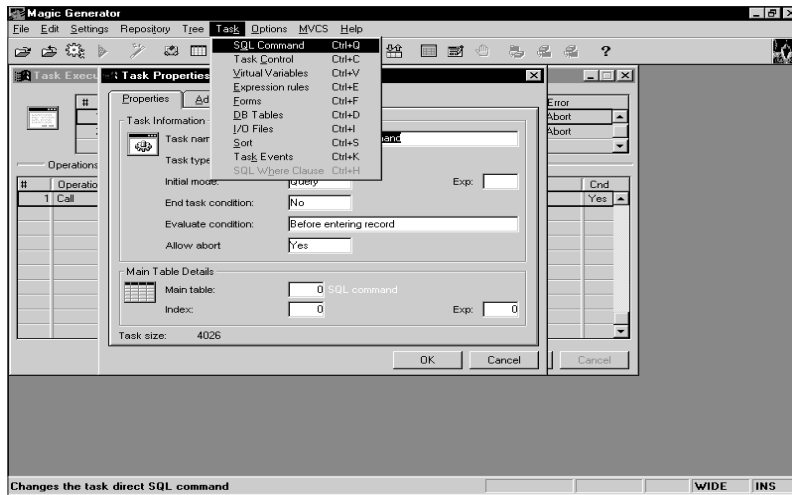


Figure 22-1 SQL Command Highlighted on Task menu



Magic executes the SQL command before it executes the Task Prefix task level. The task can use the returned data as an integral part of its dataview.

Magic does not attempt to analyze the command's syntax and semantics. This gives the programmer flexibility, but requires that the programmer plan the programming carefully. Magic does not protect the data from errors in the embedded SQL commands.

Magic's embedded SQL feature is a tuning tool to improve performance. Use this feature selectively, and only when its use results in significant performance improvement.

Use the embedded SQL feature:

- To replace a simple Link Query operation. A simple link is a link with a static link expression between tables of the same SQL database.
- When you need to calculate statistics on the database, such as how many records have a specific property, or the total sum of this month's salaries.
- When you want to make a vertical update to your data, such as to increase all salaries by 5 percent, delete all old records, and copy parts of one table to another table.
- When you want to utilize existing code that was developed and compiled using the SQL DBMS tools, such as stored procedures.

# Explicit Embedded SQL Elements

## Create an Embedded SQL Task

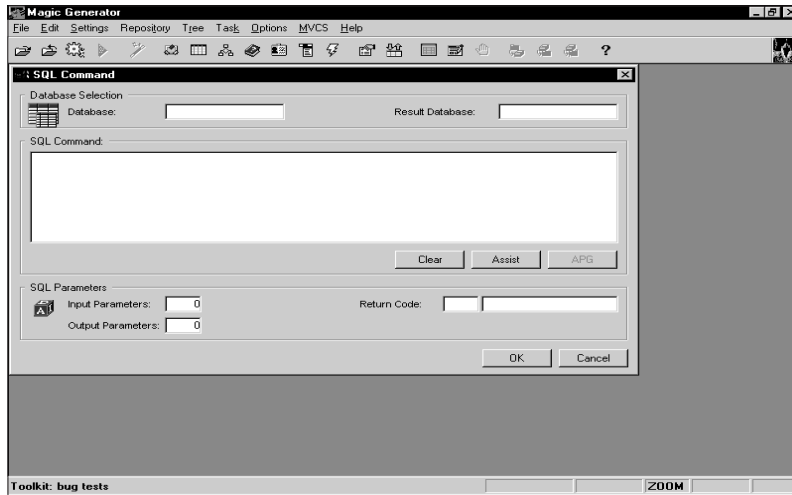


Figure 22-2 SQL Command Dialog

1. Select the SQL command from the Objects menu. The SQL Command dialog, shown in Figure 22-, appears.
2. From the Database field Zoom to the Database List.
3. Select the database from which to run the explicit SQL.

## The SQL Command

Magic does not check the syntax or semantics of the statement. The underlying database performs all statement processing at runtime. If an error occurs at that stage, the Magic task terminates with an explanatory message.

The SQL Command may contain the name of a predefined procedure that was developed and compiled using DBMS facilities (stored procedures). If such a

procedure name is specified as the SQL Command, the database procedure is invoked and executed by the RDBMS. These stored procedures can be used as though they were SQL command types. All the rules regarding embedded SQL commands described here apply to stored procedures when called.

## ***Input Parameters***

You can insert a colon (:) followed by a number anywhere in your embedded SQL command as a parameter designator. The colon plus number combination is replaced by the parameter value specified in the SQL command form's Input Parameters table. The parameters are computed just before the SQL Command is prepared, and Magic replaces the parameter designator with the computed value in the SQL statement's text. The statement is then passed to the RDBMS for syntax evaluation and execution.

The SQL command syntax is always re-evaluated before the command is executed, regardless of the Resident Task flag value. Generally, Input Parameters are used as place holders for SQL WHERE clauses.

## ***Output Parameters***

SQL SELECT statements or stored procedures provide Magic with an alternative dataview to the standard Main Table ordinarily used. When the SQL Command results produce a result table or a data stream, Magic provides the buffers necessary to accept the data. These buffers take the form of virtual variables defined in the SQL Command's task. The virtual variables must match, in attribute and picture, the data generated by the SQL Command.

## ***The Assist Utility***

The Assist utility provides an easy way to construct SQL statements that include database names for tables and columns.

The SQL Command Assist utility is not a substitute for knowing the SQL language, which is a prerequisite for using the SQL command.

Use the Assist utility to reference the names of tables and columns in the database, but be careful when you use this utility to build syntax.

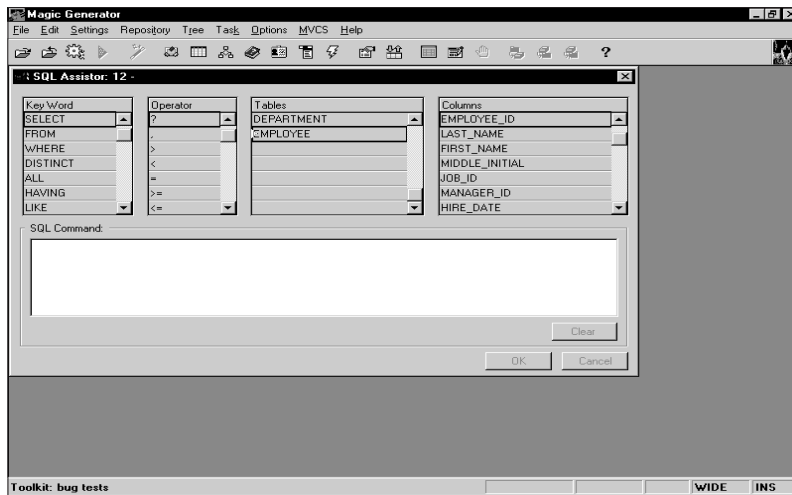
For example, the Assist utility will build

```
SELECT ALL FROM EMPLOYEE
```

even though this is not valid syntax. The correct syntax is:

```
SELECT * FROM EMPLOYEE
```

## *Use the Assist Utility*



*Figure 22-3 SQL Assistor Dialog*

1. Click the Assist button on the SQL Command dialog. The SQL Assistor dialog, shown in Figure 22-, appears.
2. Click the Key Word button or type CTRL+K to put the cursor in the Key Word box.
3. With the up and down arrows, move to the SELECT statement.
4. Press ENTER to place the word SELECT in the Statement box.

5. Click the Oper button or type CTRL+O to put the cursor in the Operator box.
6. With the up and down arrows, move to the \* operator.
7. Press ENTER to add the \* and a blank to the existing text in the Statement box.
8. In the Statement box, type a space and the word FROM.
9. Click the Table button.
10. With the up and down arrows, move to the employee table.
11. Press ENTER to place the employee table in the statement box.

You can view the table under either its Magic name or its RDBMS name. By clicking the Flip button, you can toggle back and forth between the Magic and RDBMS table names. Whether selected from the Magic or the RDBMS name, the SQL statement shows the RDBMS name.

Note: The Table box shows only tables that are defined in Magic's File Dictionary. However, it is possible to enter SELECT statements that access any available tables in the database, even if those tables are not defined in the File Dictionary.

## ***The SQL Command Automatic Program Generator - APG***

The SQL APG utility constructs a complete Magic program from SQL statements the programmer inputs. Running the SQL APG is the only way to check the SELECT statement syntax before running it. The utility generates a Magic task structure based on a SELECT statement or a stored procedure.

The generated task contains the following items:

- The original SQL Command
- Select virtual operations and virtual field definitions for all the result table's columns
- A full Output Parameters table with the automatic virtual variables

- A default form for user interaction

The developer only has to define input parameters for the SQL command if required and change the default form if necessary.

### ***Use the APG Utility***

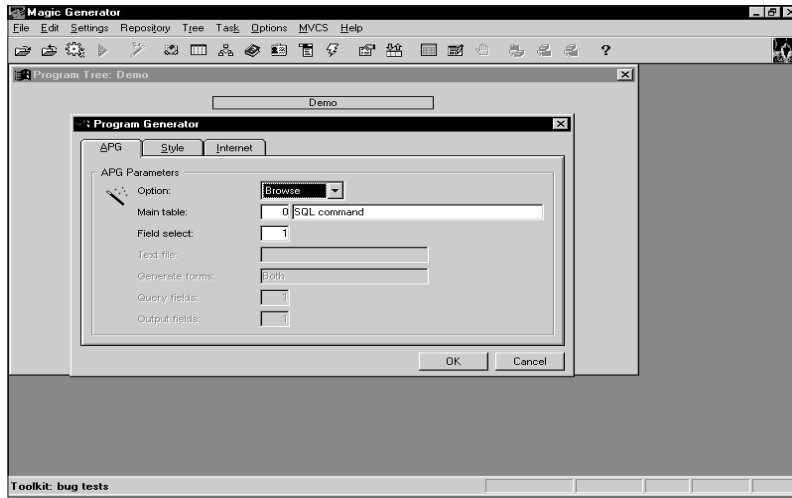


Figure 22-4 Program Generator Dialog

1. Click the APG button in the SQL Command dialog. The Program Generator dialog, shown in Figure 22-, appears.
2. Press ENTER twice.
3. Press F7 to run the program.
4. Enter the Output Parameters Field and Zoom to the Output Parameters created by the APG utility. The Output Parameters are related to the virtual fields in the Variable List.

### ***Look at the Program Created***

1. Type SHIFT+F3 to access the Program Dictionary.

2. Bring the cursor to the Demo program you built by Zooming to the Program Tree.

As you can see, all the expressions returned from the SELECT statement appear in virtual fields and are displayed in the generated program.

## ***Behavior of Explicit SELECT Statements***

Some of the direct SQL statements that have a result database behave differently in Magic.

Note: Direct SQL statement execution always occurs before a task prefix. Therefore, once you have entered the task prefix, the SQL statement has already been executed.

### ***Online vs. Batch***

In online tasks:

- You must browse on the SELECT statement result.
- Magic creates a temporary result table/file and inserts all the records into it so that the user can scroll on the records. The table is deleted at the end of the task.

In batch tasks:

- Every record is only read once, and you do not need to go backwards.
- Magic opens a cursor according to the SELECT statement and retrieves all the records one after the other.

### ***Result Database as Input Database***

Selecting the Result Database as Input Database option lets you make the result database the same as the input database. In most cases, the underlying RDBMS allows an

```
INSERT INTO table AS SELECT...
```

statement, which copies all the data to a table in one command. This method is faster than opening a cursor. All of the records are retrieved from the client and inserted into the result table.

In such cases, the SQL gateway creates a table in the database and sends:

```
INSERT INTO temp_table AS SELECT "direct SELECT  
statement"
```

The user may then scroll on that table to speed up the SELECT statement execution. This method is especially helpful when the result set is large because there is no retrieving and inserting of each record.

## ***Result Database Different from Input Database***

When the result database is different from the input database, a temporary table is created. A cursor is defined, and the records are retrieved and inserted one by one into the result table. The user may then scroll on that table.

## ***Recommendations***

- When the result is large, it is best to use the option that makes the result database the same as the input database. Then the INSERT INTO AS SELECT statement will be used.
- When the result is relatively small, it is best to use the result database as an ISAM database that will reside on the client. Then when you create the result and scroll on it, the work is only performed on the client, which reduces network traffic.
- Use the memory gateway when the results are relatively small to enhance performance.



## ***Restrictions on Using Embedded SQL***

A task with an embedded SQL command instead of a main table is a normal Magic task and has almost the same functionality. There are, however, some restrictions:

- All the tables participating in the SQL command must be from the database that is declared in the Database column or the SQL command properties. Mixing tables from different databases is not allowed.
- In an online task, do not update a column that belongs to the SQL command's view (the result table). Such updates are never written back to disk because the result table is deleted when the task terminates. Use Links to other tables to update information.
- Do not use commands that alter the state of a record that is the current view of an ancestor task.
- During batch processing do not update an On Change Level of a SELECT operation. On Change operations require that the preceding record be refreshed before the On Change operation. Refreshing records is impossible because the task's data stream is a one-way stream that cannot be stopped.
- Use task level transactions on batch SQL commands.
- Do not use the COMMIT and ROLLBACK commands in your embedded SQL. If these commands are not generated through Magic's standard transaction management layers, the results are unpredictable. COMMIT and ROLLBACK functionality is achieved correctly only by choosing the right transaction mode in Magic's task level table.
- The SQL statement is executed by the underlying database. The command syntax is the developer's responsibility, and the developer is not prevented from using DBMS-specific extensions. If application portability among various SQL DBMSs is required, be careful not to include DBMS-specific SQL extensions in embedded SQL. In the SQL statements created by Magic this is not a problem. The Magic gateways generate the correct, optimized syntax for each database.

- When using embedded SQL in online tasks, Magic creates a temporary table. Optionally, the table can be created in the database. Creating the temporary table in the same RDBMS using that RDBMS's utilities is usually more efficient than having Magic read and write each record.
- Range on DSQL output parameter is not allowed.

To execute a stored procedure, prefix the name with the word `exec` as in:

```
exec sp_order_update
```

The gateway actually executes the generated program statements to recognize the correct form of the result set in the APG. So, if your statement modifies data, the modification will occur.

## ***Error Handling***

### ***The DBERR Function***

Because SQL commands are not analyzed during development and the final syntax is only known at runtime, it is good practice to check the value of the `DBERR` function in the parent task immediately after the task is called with the SQL command. This indicates to the programmer whether or not the SQL command was successful.

---

**N**ow that you understand how Magic & SQL work, it is important to learn how to work with Magic & SQL efficiently. Various concepts and techniques for achieving optimal Magic & SQL performance are discussed below.

## *Key Definition and Usage*

For best response time RDBMS indexes should be used for most data retrieval. Usually the RDBMS uses one of the indexes when the SQL statement has a WHERE clause constraint on one or more first segments of that index and the requested record order is consistent with that index.

Each of the following examples illustrates which SQL statements use the indexes and when. For these examples, assume there is an index IN1 on fields F1, F3, F5 of table TBL1, and another index IN2 on F3, F4, F5. Magic issues this statement when using the first key and ranging on F1 with the same expression for FROM and TO, and on F2 with two different expressions.

**Example:**

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
WHERE F1=1
AND F3=100
AND F3=200
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1 and F3. The order will be achieved automatically by using the index.

Magic issues this statement when using the first key and ranging on F1 with the same expression for FROM and TO, and on field F2 with two different expressions.

**Example:**

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
WHERE F1=1
AND F2<='x'
AND F2>='c'
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1. The RDBMS searches all records with F1=1. Only those with F2 between *c* and *x* will be in the result table. The order will be achieved automatically by using the index.

Magic issues this statement when using the first key and ranging on F1 with the same expression for FROM and TO, and on field F3 with two different expressions.

**Example :**

```
SELECT F1, F2, F3, F4, F5
FROM tbl1
WHERE F1>=1
AND F1=10
AND F3>=100
AND F3<=200
ORDER BY F1, F3, F5
```

Index IN1 will be used for the range on F1. The RDBMS searches all records with F1 between 1 and 10 and compares them to the range of F3 values. The range on F3 is not done by using the index because the range of the previous segment was not on a single value. The order will be achieved automatically by using the index.

Magic issues this statement when ranging on F1 with a single expression and on F3 with two different expressions and using the second key.

**Example:**

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
WHERE F1=1
AND F3>=100
AND F3<=200
ORDER BY F3, F4, F5
```

Index IN1 will be used for the range on F1 and F3. The RDBMS orders query results by sorting the result table. The second index cannot be used to supply the requested order because the first index is used for range. Using sort is relatively fast, as only the result table will be sorted and in most cases it will be relatively small.

Magic issues this statement when using the first key and ranging on F3 with two different expressions. The second index will be used to range on F3 and the result is then sorted without an index.

**Example :**

```
SELECT F1, F2, F3, F4, F5
FROM TBL1
ORDER BY F1 DESC, F3 DESC, F5 DESC
```

The RDBMS performs a full sort on the entire table. The only way to avoid a sort in this case is to define another key on the same column as the first, but in descending order.

**Note:** Indexes take up space in the database and are time-consuming when inserting, updating, and deleting records from the database. In some extreme cases indexes can cause poor performance for SELECT statements. For example, when a table's data comprises less than a block of disk space, which can be several thousand records for a normal table, a SELECT statement does not perform well. When accessing the table through an index the RDBMS actually executes two I/Os; one for the index and one for the data. Executing a full table scan on a single block, which was read into memory in a single I/O, is faster because memory access is always faster than disk I/O. When all of the columns selected in the query are in the index, the RDBMS will then access only the Index and not the data - this is called a cover index query.

However, in most cases, indexes will enhance the speed of your application.

## ***Range Definition***

When browsing large tables in relational databases it is important to use ranges to reduce the number of records in the view. When a table is accessed without ranges, such as in the APG, some of the operations can take a long time, especially operations such as locate and page up. To improve performance the ranges should be on segments of an index.

Magic lets you specify ranges from two places:

- Magic SELECT statement - All ranges mentioned in the Magic SELECT statement become part of the SELECT statement. The range will then be handled by the RDBMS with a WHERE clause, even during a sequential search. Magic sees only the records that answer the query.
- Range expression at the task level - When using a range expression at task level, Magic checks each record returned from the database against the expression and decides if the record is part of the view.

These two options perform very differently. It is better to put as much of the range information as possible in the range expressions at the select level. Only enter ranges that cannot be expressed otherwise at the task level. For example, if you want to select the records that have values from 100 to 200, or from 300 to 400 in fld1, you cannot define this range at the select level. Instead,

define a range expression at the task level. You can improve performance by adding a range from 100 to 400 at the select level.

## ***Transactions***

Using transactions explicitly helps achieve data integrity and good performance. In general the longer the transactions the better the performance. On the other hand, long transactions in online tasks can cause locking problems. General guidelines for using transactions on both online and batch processing follow below.

### ***Online Tasks***

For online tasks, use one of two strategies:

1. By default, transactions are used at the record level. Before the update stage in the Magic cycle, Magic opens a read/write transaction, performs the updates, and commits the transaction. This strategy is compatible with Magic's normal behavior using ISAM files. However, it is impossible to maintain referential integrity in this way because each line in line mode is inserted or updated when leaving the line.
2. Execute all writing to the database in a separate batch transaction after all the transaction's data entry is complete. During data entry temporary files should hold the data. This method maintains referential integrity and avoids locking problems. Extra complexity is the main disadvantage.

### ***Batch Tasks***

In batch processing it is important to minimize the number of transactions by telling Magic to start a transaction at the task level. If the task is very long, a transaction can be opened at a change level. If the batch task does not reach the commit point, all the writing to the database will be automatically rolled back. When committing at a change level, there must be a way to recognize

which part of the task completed. This can be done by setting the task transaction to YES in the highest level batch task in the tree, which causes the whole task and its subtasks to be in one transaction.

For example, there is a batch task that creates monthly invoices for customers based on the deliveries recorded for each customer. If all customers are processed in one transaction and the task fails before completion, the task has to be restarted for all of the customers. If a transaction is opened at the new customer level, a control record can maintain the last customer processed. In the event of failure, only the customers that were not processed before the task failure need to be processed.

When updating all or most of the records in a table, it is best to open the table in exclusive mode (write/none) so that only one lock is held at the table level, rather than locking each record.

## ***Sorting***

Internal sorting in Magic is very costly. It is always best to ask the RDBMS to sort the data and supply the records in the required order to Magic. This is done by defining keys in the Magic File dictionary.

Keys can be defined as virtual, which means that no RDBMS index is defined. The virtual key causes Magic to create an ORDER BY clause for the SELECT statement being sent to the database. Magic uses virtual keys in the same way as real keys, but does not try to create an index in the database based on this key. Virtual keys should be used in main files (typically in batch tasks) and not for linking.

## ***Embedded or Explicit SQL***

Embedded SQL can provide better performance by using RDBMS features that are not used in normal Magic programming.



Global update to table rows and global delete of table rows are good examples. It is especially important to use global statements when working in a Client/Server environment to avoid excessive network traffic.

Another example is using statistical functions such SUM, TOTAL, or AVG. If you want a record count, it is more efficient to use SELECT COUNT(\*) in embedded SQL than to read records just to count them.

## Views

Using RDBMS views instead of simple tables can improve performance when joining tables. Magic's link operation is implemented by issuing a separate SELECT statement to the database for every linked table. Batch tasks can be performed more efficiently by using a view that joins the main file and the linked files. Let the database do the join. Then have Magic use the view as the main file, and issue only one SELECT statement for the task.

Note the differences in how Magic and RDBMSs use the term view:

- Magic uses the term view to describe the fields selected from the main file together with fields selected from the linked files and virtual fields calculated by the real fields.
- SQL databases use the term view to describe a pre-defined SELECT statement saved in the internal Data Dictionary and used as a table. The view definition can join several tables, use statistical functions, and use a WHERE clause to select part of the records in the table.

### Example:

There is a task with a main file containing 10,000 records and three linked files. If performed in the usual way, the number of SELECT statements executed by the task would be:  $1 + 3 * 10,000 = 30,001$

If you define a view in the database, the number of SELECT statements executed by the task would be: 1

When using the SELECT statement technique, remember that joins defined in a view are inner joins by default, and Magic links are outer joins.

**Example:**

Join table F1 to table F2 based on column C3 in table F1.

To do this in Magic, all records in F1 will be in the Magic view. In records where column C3 is empty, or where the value of C3 does not match a record in F2, the fields from F2 are empty.

Using the RDBMS's inner join, only records in F1 that match records in F2 appear in the Magic view.

## ***Stored Procedures and Triggers***

Stored procedures provide a very powerful way to move parts of the application logic to the server. They can be called from within the Magic environment by using the Direct SQL feature.

In Sybase, stored procedures can receive parameters and can create a result table similar to SELECT statements.

The syntax of stored procedures differs by database, so using this feature may conflict with the application's portability requirements. For more information, see the section on Application Migration and Portability.

Triggers are automatically invoked when records are inserted, updated, or deleted. When using triggers with Magic, make sure you know which triggers already exist to avoid doing the same operations from the Magic programs.

## ***Client/Server***

The best way to implement Client/Server architecture is to have each machine perform the part of the application it is best equipped to handle. Usually the client supports the front end, and the server deals with the database. It is necessary to divide the task between the client and the server so that large amounts of data can pass between the client and the server over the network. Reducing network traffic is the best way to improve performance.

Follow these guidelines to reduce traffic on the network:

- Execute batch jobs on the server. Avoid executing large batch jobs from clients.
- Use views instead of linking when possible.
- Select only the fields you really need from the record. Only those fields will be transferred In SQL.

[This page intentionally left blank]

---

**W**hile not automatic, transitions from one DBMS to another are relatively easy in Magic and SQL. The procedures to be followed for different types of application migration are described below.

## ***What's New in Magic 8***

Many SQL enhancements are contained in Magic. They can be categorized as follows:

- New Operations—Link Join, SQL Where, Sort Using RDBMS, and ON Lock Transaction mode.
- Embedded SQL Flags from Magic 6/7 into Magic 8—Positions, Hints, SQL Types, Isolation, etc.
- New Behavior of Magic 8—Default Mechanism, Cache Strategy, and DB Name Automatic Generation.
- New Functions—INTRANS, CNDRANGE, and DBDISCNT.
- New Terms—ISAM terms have been changed to SQL terms.

## ***Migrating from Magic 6 or 7 to Magic 8***

Mapping Magic attributes to the RDBMS datatypes in Magic 8 is easier than in previous versions. The Mapping rules for all attributes are presented below:

1. All RDBMS data types that map to Magic alpha are stored internally as Zstring. When importing an application from Magic 6 or 7, all SQL-based alpha columns have Zstring storage. There is no effect on the physical data.
2. Magic supports up to 18-digit numbers. In SQL gateways when the column is larger than 15 digits it is stored internally as a Magic number. This does not affect the physical data.
3. All Magic Date attributes are mapped to a Date String, and the length of the string is only 8 digits. It is not necessary to change the database.

When importing an application from Magic 6 or 7, all SQL-based Date columns will have the length of 8.

**Note:** If CHAR (6) was used as the SQL type in Magic 6 or 7, the data must be changed.

4. The Informix default storage for Time is String Time, as opposed to Integer Time in Magic 6 or 7. No change in the application is required.
5. The Informix default SQL type for String Time is 'DATETIME HOUR TO SECOND' instead of CHAR (8). In the event that String Time is mapped to CHAR in the Database in Magic 6 or 7 applications, simply add the SQL type-CHAR (8), to that column.
6. All Magic Memo attributes have a Magic length that is the maximum length. However, in Magic 8 Magic stores only the physical length of the column in the database and does not fill it with blanks.
7. Magic Memo attributes in the Informix database default mapping have changed to VARCHAR when the length is 255, and to TEXT when the length is >255.  
In the event that Magic Memo is mapped to CHAR in Informix, in Magic 6 or 7 applications simply add the SQL type CHAR(n) to that column.

8. When importing an application from Magic 6 or 7, Magic will add the SQL type CHAR to all Memo columns in Informix.
9. The Oracle default SQL type for String Time is CHAR, instead of RAW in Magic 7.

## **ISAM – RDBMS**

Writing an application designed to work in an ISAM environment (usually Btrieve or CISAM files) is different from writing an application when the data resides on an SQL database. Some elements are not handled or taken into consideration when working with ISAM files. Therefore, some parts of the application may be easily changed while other parts may require more programming.

When moving from an ISAM-based application to an RDBMS-based application, you should follow the guidelines listed below:

- General settings - The general settings in both the MAGIC.INI file and in the Environment table should be changed, as described earlier in the section on Defining the Magic Environment.
- Table repository - The first thing to be changed is the Table repository. This can usually be done easily using a script for exporting and changing the Table repository.

First you change the database for each table. As a result, the field's storage type is automatically changed to the default storage type of that specific SQL gateway attribute. You may keep the data types in the default, but be sure that Date columns that use Zero date are used with the SQL type property as `char ( 6 / 8 )`.

It is best to add the dbname to each column because you cannot specify the dbname in ISAM.

Nulls may be added when necessary.

- **CONVERT** - When you change the database for each table and the Change Table in Toolkit option is set to Yes, Magic tries to convert the

data for you. You can convert a small amount of data in Magic, but to load the data from a large database, export the tables to an ASCII file and use the RDBMS tools such as SQLloader and bcp. These tools are faster and more reliable than the Magic CONVERT operation.

- **Select all columns with no default values** - When a column does not allow nulls, the column must appear in the INSERT statement. Therefore, in a task that inserts records, you should select all the non-nullable columns. Magic will create the INSERT statement according to the columns selected in the task.
- **Select all position columns** - When using logical locks in RDBMSs such as MS-SQL and Sybase, Magic needs to know the values of all the segments of the unique index used as the DBPOS, as described in Chapter 19. Therefore, in a task which may update or delete records, you should select all the columns which are part of the index segments.
- **Indexes** - Non-unique indexes are commonly used in ISAM tables. All the indexes should be changed to unique and made one-way keys if possible.
- **Indexes** - The indexes in the database should match the indexes defined in Magic so that minimum sorting is performed in the application.



- **Links** - A link operation is often used in applications written for ISAM. The Link operation requires a lot of resources and therefore causes the task to run more slowly.

These links should be reduced to the minimum, which can be done by:

- Defining a View in the database that joins all the relevant tables
  - Using the cache to reduce the number of times the records are retrieved from the database
  - Performing column or code validation in a separate program and calling this validation only when needed
  - Mark the table as Resident if it is not large and no updates have been made to it.
  - Use the Link Join operation where applicable.
- **Transactions** - It is common for transactions not to be handled at all in ISAM environments. However, in the RDBMS environment everything is performed by a transaction. If the transactions are not changed in Magic, the duration of locks may be shorter than required, or a database error may result in a ROLLBACK of the whole operation. Therefore the transactions should be scanned and changed where needed. In most cases the changes may not be big, and then the Magic default options may be acceptable.
  - **Adding Direct SQL Stored Procedures** - To make the application run faster and reduce network traffic between the client and the SQL server, you may also use direct SQL statements for vertical updates or for a complex Range in addition to using stored procedures.

This change may result in major performance improvements. However, the procedure is quite advanced and should only be performed by someone with good SQL skills who understands the application.

- **Magic locking** - When using the RDBMS locking mechanism, Magic locks may be omitted in most cases to improve performance. The locking behavior is the same as in other SQL applications. In some cases you

must verify the behavior or leave the Magic locks on for a few files, especially when logical locks are issued and a page level lock is used.

## ***RDBMS – ISAM***

Converting RDBMS-based applications to ISAM-based applications is easier than the other way around. Problems sometimes occur with programs that worked in SQL only, such as views, virtual indexes, direct SQL, nulls, stored procedures, and with any logic that was built into the database and now has to be implemented in the application.

Transactions may be ignored. The use of unique one-way indexes and of the Magic cache may be kept and will enhance performance.

## ***RDBMS – RDBMS***

An application written to work with one RDBMS may be required to work with another RDBMS. This adaptation is simpler than the ISAM $\leftrightarrow$ RDBMS transformation.

In most cases, all you need to do is change the database in the Table Repository for each table, thereby changing the storage types. However, sometimes more is required.

- Syntax of a direct SQL statement may be valid in one RDBMS but invalid in another. When this is the case, the syntax must be changed.
- Stored procedures must be rewritten.
- If you kept the definition of views as a magic program, you can simply run them in the new RDBMS. If not, views must be rewritten.
- When moving between a physical, row-level-locking RDBMS (such as Oracle, Informix, or DB2) and a logical, page-level-locking RDBMS (such as MS-SQL or Sybase), additional testing is required because the behavior might change.

In general the RDBMS ↔ RDBMS transition is easy, and in most cases no program changes are required.

## ***Keeping the Lowest Common Denominator***

Sometimes the same application needs to run in both the RDBMS and the ISAM environments. In this case, the programmer only needs to change the Table Repository by using a script that scans and changes the CTL export. The programs should not be touched.

One way to ensure that the application will run in both environments is to keep the lowest common denominator that works with both SQL and ISAM, as described in the RDBMS → ISAM section. The use of views, direct SQL, and virtual indexes is handled by multiplying the programs, and a switch tells the application if it is running in an ISAM environment or an SQL environment. The application decides which program to use in runtime.

The use of direct SQL views and virtual indexes may be limited only to the essential locations in the application.

In runtime, the application may decide which DBMS is used by the INIGET function on the database section in the Magic INI file. The application will then use the correct program for this RDBMS, so that a view will be used in the RDBMS and links will be used in ISAM.

You should use the underlying RDBMS tools and features that were designed to do specific tasks. It is not advisable to treat the underlying RDBMS as an ISAM table system.

[This page intentionally left blank]

**T**he follow chapter provides general technical information for all SQL databases.

## ***Magic SQL Command Object***

You can define any SQL statement in the SQL command object, using the restrictions that are listed below.

### ***General***

You can use the Magic SQL command object to enhance performance or use advanced features of SQL that are not automatically used by Magic.

### ***Select Statements***

You can use the Magic SQL command object to perform any SQL command, execute stored procedures, or use such SQL features as Group By, Sum or Distinct.

**Note:** The gateway uses a cursor for the Select statement. Only valid server Select statements using the server cursor can be selected.

## ***Stored Procedure***

You can use the Sybase stored procedures by specifying the reserved word `exec` in the following format:

```
exec procedure-name parameters
```

If the procedure accepts more than one parameter, separate the parameters with commas. If the procedure body is a `Select` statement, treat the procedure as if it is a regular `Select` statement. In other words, select *Options/Generate Program*, or select the APG button. There are three APG characteristics that should be kept in mind when in a stored procedure:

- APG invokes the stored procedure. If the stored procedure contains statements other than the `Select` statement, change the other statements into `Remarks` while using the APG.
- Stored procedures that are called from Magic and receive parameters by value.
- An output parameter can only result from a `Select` statement.

**Note:** If the procedure body contains more than one `SQL` statement, only the first `SQL` statement will be processed. The gateway opens an extra connection for each database server where you will use that server's stored procedures.

## ***Result Database***

If a Result Database is defined as a Sybase database in the `SQL` command form, it must be on the same database server as the source database. In the case of DB2 and ODBC, the Result Database cannot be the source database.

## ***Automatic Program Generator***

Magic's APG (Program Generator) is used to create simple Magic programs that can handle a result set, the output of a `Select` statement.

## ***Other Statements***

You can use the Magic SQL command object to perform global updates, global deletes, DDL statements, and PL/SQL Blocks.

## ***General Issues***

### ***SQL DATETOALPHA Parameter***

The MagicGate Database Gateway supports the use of the SQL\_DATETOALPHA parameter at the Database level.

The SQL\_DATETOALPHA setting automatically converts the RDBMS date field to a Magic alpha zstring field with a length of 19 characters and with the format YYYY-MM-DD HH:MM:SS.

**Note:** Neither Magic nor the RDBMS can perform data validation on an Alpha\_Date field due to the use of the internal date format in the RDBMS. If you use the SQL\_DATETOALPHA parameter you should implement your own validity checks for Insert and Update operations. Otherwise, invalid dates can be inserted in the database.

## ***Security***

Magic has its own security mechanism and it precedes the RDBMS security mechanism. Any restrictions imposed by the Magic security system will automatically apply to the RDBMS actions. Any security rules defined in the RDBMS will apply restrictions in addition to those applied by the Magic security mechanism. For additional information see *The Magic Guide to Application Development*.

## **Error Handling**

Whenever the gateway fails to perform a Magic request, Magic displays an error message describing the operation that failed. To process the error's return information, you can use the DBERR function in your application. For additional information about the DBERR function, see the Functions chapter in *The Magic Reference*.

## **Sort/Temporary Database**

The ISAM database is recommended for a Sort/Temporary database. Other databases create a temporary table in Sybase and in the SQL Server for the Sort/Temporary table. This temporary table can cause a transaction failure.

## **Triggers**

Triggers are transparent to Magic. Although Magic is unaware of the existence of triggers, it will work with any trigger. If a trigger fails, the Magic action that invoked the trigger will cause an error.

## **Rules**

Rules defined in the database are transparent to Magic. Magic does not know about the existence of rules, but will work with any rule. If a rule fails, the Magic action that caused the rule to fail will cause an error. The databases that support Rules are MS-SQL, Sybase, and Informix.



# Oracle

The following functions are available for the Oracle database.

## **Optimizer Hints**

Using a Hint String, the developer can specify a hard-coded string that can be added to the Select statement as is, without being checked.

The syntax of a Hint is :

```
/*+Oracle Hint */.
```

It is recommended that you use the Optimizer Hints in special cases only. You can read about the hints in the underlying RDBMS documentation before using them.

## **Stored Procedure**

It is possible to use the Oracle Stored Procedures by specifying the reserved word **exec** or **execute** in the following format:

```
exec procedure-name<parameters>
```

or

```
execute procedure-name<parameters>
```

Procedures without INOUT or OUT parameters must be called as a PL/SQL block:

```
begin procname (par1 int,par2 int); end;
```

If the procedure accepts more than one parameter, separate the parameters with commas. If the Oracle parameters are **inout** or **out**, the APG button must be selected before executing the procedure. Do not enter an **out** parameter in a Stored Procedure. Instead, use a comma as a place-holder. Do not add any punctuation at the end of the procedure.

**Notes:** Procedures with **inout** or **out** parameters work in batch mode or in online mode when the Result database is not Oracle. The date format in the **inout** parameters must be YYYYMMDD.

## ***Multi-Connections***

Oracle's Database Link feature enables you to use many databases located on different servers in a single Magic session.

## ***Views***

A View must have a virtual unique index defined. Insert, Update, and Delete operations are allowed on Views. A View that is defined on more than one table does not have a ROWID. Therefore, the Position parameter on the SQL tab in the Table Properties dialog should be Unique Index, and not Default or Rowid.

**Note:** Magic relates to View as a regular table. It is recommended that you not use Magic to perform any type of rename or convert operations. If you do execute one of these operations, Magic will display an error message and will not convert/rename the View in the database.

## ***Unique Views***

Magic must have a unique row identifier for each file that it opens. Whenever possible, Magic will use Oracle7's ROWID as the unique identifier. If the dataview is built from fields from more than one Oracle database table, there will be no Oracle ROWID, and a unique index must be entered. This may be a virtual index.

## ***Two Phase Commit***

If you want to support Two Phase Commit operations with multiple connections in Oracle, you must use Oracle's Database Link feature.

## ***Create Table Parameters***

It is possible to specify any of the following parameters in the Table Properties dialog, when you are creating a table in Oracle:

TABLESPACE=...

INITRANS=...

MAXTRANS=...

PCTFREE=...

PCTUSED=...

CLUSTER=...

STORAGE=...

## ***Load Definition from a Remote Server***

Magic enables you to use an Oracle database on a remote server that is connected to a local host using Oracle's Database Link feature, which is represented by a synonym on the local host.

It is necessary to first load the database definition from the remote server, and then enter the synonym location in Magic's Table repository. The following steps should be performed:

1. In the Database column of the Table repository, enter the name of the remote database.
2. In the DB Table column, enter the name of the table or view in the remote database.
3. Execute the Magic Get Definition procedure.
4. Change the database name to the name of the local database containing the synonym for the linked database.

## ***Deferred Execution of SQL Steps***

Oracle7 enables you to defer the execution of one or more steps in the processing of an SQL statement. Deferred execution can result in enhanced performance, but it does use more client memory. This gateway version introduces a new Magic DBMS parameter that enables you to specify your preference for deferred execution. The parameter is called NO\_DEFER, and the default is NO\_DEFER=N, meaning that execution is deferred. If available client memory is restricted, set NO\_DEFER=Y, which will cause execution of each database step. If available client memory is restricted, set NO\_DEFER=Y, which will cause each database to be parsed and bound as it occurs.

## ***NLSSORT Support***

The NLSSORT Support feature lets the application match character strings that follow alphabetic conventions. Normally, character strings in a WHERE clause are compared by using the character's binary values.

Using the NLSSORT Support feature in the WHERE clause allows a user to work on a foreign-language Oracle client while sorting data alphabetically in their own language.

To use this feature, add the following flag in the *Database Properties/SQL/Database Information* field in the Database repository:

```
NLSSORT=Y
```

This flag adds the NLSSORT function to all the WHERE clauses that Magic sends to the database, as follows:

```
NLSSORT (value) comparison_operator NLSSORT(column)
```

Note that use of this flag may cause performance problems. Use of this flag also causes the Magic Oracle gateway to be not case sensitive.

# MS-SQL

The following functions are available for the MS-SQL drivers:

## ***Optimizer Hints***

Using a Hint String, the developer can specify a hard-coded string that will be added to the Select statement as is, without being checked. When the hint (HOLDLOCK) is specified in the Index properties, the gateway generates the command:

```
Select fld1 from table1 (HOLDLOCK) order by fld1 asc.
```

Specifying FORCE\_INDEX Hint in the Index Properties dialog applies to this index only. Specifying FORCE\_INDEX Hint in the Table Properties dialog applies to all the indexes in that table, and specifying FORCE\_INDEX Hint in the Database Properties dialog applies to all the indexes in all the tables in the database.

If you specify a FORCE\_INDEX Hint in the Table or Index Properties dialog, and you do not want the gateway to force the optimizer to use an index for a specific index, or for all indexes in a particular table, then you can set the Hint to NO in the Table or Index Properties dialog, and the optimizer hint will not be issued.

FORCE\_INDEX Hint can cause the gateway to force the index in the following manner:

Add the following to the Select statement generated by the gateway:

```
"(INDEX indname)"
```

Indame will be the name of the index for that particular table in Magic.

It is recommended that you use the Optimizer Hints in special cases only. In addition, the Hints description in the underlying RDBMS documentation should be read before use.

## ***Multi-Connections***

Multiple databases can be used within a single Magic session even though the databases are located on different servers. The MS-SQL gateway opens one connection for each new server-user pair to which you are connected. Specifically, connections to the same server but with different user names result in multiple connections. The MS-SQL gateway opens an extra connection.

## ***Views***

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted on Views.

MS-SQL allows updates on multiple table views. If one of the segments of the Position Index is updated, records shown on the screen may become inconsistent. It is important to note that Magic considers a View as a regular table. Magic should not be used to perform any rename or convert operations. If Magic is used to execute any rename or convert operations, Magic will display an error message and will not convert/rename the View in the database.

## ***Locking***

No explicit table locks exist in MS-SQL. Therefore, Share-None and Share-Read have no effect. Locking will be carried out by the SQL engine.

## ***DB Commands***

The gateway uses cursors by default. When processing a large number of records in a Magic task, such as in a batch task that scans a file, it is possible to change the default setting by changing the Cursor parameter in the Table Properties dialog to No. This will cause the gateway to use DB commands instead of cursors, requiring separate connections for each result set. The

maximum number of connections is user-defined, and the default setting for the maximum number of connections is 3.

The gateway uses one additional connection for commands that execute a single record or command, without fetchable results such as “Update”. The gateway uses other connections for commands with fetchable results.

Generally speaking, the greater number of connections the gateway uses, the better the client performance. However, at the same time memory requirements increase while server performance decreases. If all of the existing connections are already used by a pending command and a new connection is needed, an existing connection will be freed for your use according to the LRU algorithm. This can affect performance adversely, because the released command will eventually be reissued.

If a Direct SQL batch task is used (Stored Procedure or Select statement), the gateway is unable to reuse this connection until all the results have been fetched. If nested direct SQL batch tasks are used, and the maximum number of connections the gateway can use is not enough, the following error message will appear:

**MS-SQL Gateway: No more connections available. Try increasing Max Connections in the DBMS properties.**

If this error message appears, either increase the value of the parameter noted in the error message or modify your application, so that it does not use the nested Direct SQL tasks.

## ***Relevant Parameters***

- Setting the Table Properties dialog, SQL tab, Cursor parameter to No disables the use of cursors on the specific table, and uses DB command instead.
- Setting the DBMS Properties dialog, Max Connections parameter controls the maximum number of connections the gateway can use. The default value is 3 (plus 1 extra) connection. The extra connection is used by the gateway for commands that fetch a single record, or for a command without fetchable results, such as Update. This number is for the

Server/User-name pairs. If you have defined several databases in *Settings/Databases*, and some of them use different servers or different user-names, this number applies to each database that has a different server or user-name.

- A new parameter in the Database Information field of the Database Properties dialog,

`SQLBLOB=n`

allows you to specify *n*, the size in bytes of the largest blob in a table. The default value is 65534. The maximum value is 2147483647. The value of SQLBLOB is relevant only in Direct SQL and Convert in the following two cases:

- when executing a stored procedure with blobs from Direct SQL, or
- when using a direct SQL statement with a Result Database in Btrieve, or when converting a file to Btrieve, Btrieve has a limitation of 65534 for each blob. You can reduce the value of SQLBLOB to below 65534, but any attempt to increase it above 65534 will have no effect.



# Sybase

The following functions are available for the Sybase database.

## ***Optimizer Hints***

It is possible for the developer to specify a hard-coded string that can be added to the Select statement as is, without checking, by using a Hint String.

Specifying the FORCE\_INDEX Hint in the Index Properties dialog applies to this index only; Specifying the FORCE\_INDEX Hint in the Table Properties dialog applies to all the indexes in that table; and specifying the FORCE\_INDEX Hint in the Database Properties dialog, applies to all the indexes that are contained in all the tables, of the database .

If a FORCE\_INDEX Hint is specified in the Database Properties dialog, and you do not want to force the optimizer to use an index for a particular index, or for all indexes in a specific table, you can set the Hint to No in the Index or Table Properties dialog, and the optimizer hint will not be issued.

The FORCE\_INDEX Hint will cause the gateway to force the index as follows:

- Sybase 10:  
Magic will add the string (indid) to the generated Select statement. Indid stands for index id, and will be retrieved from the system table according to the name of that table in Magic.
- Sybase 11:  
Magic will add the string (INDEX indame) to the generated Select statement. Indame refers to the name of the index of that table in Magic.

It is recommended that you use the Optimizer Hints in special cases only. In addition, read about the Hints in the underlying RDBMS documentation prior to use.

## ***Locking***

There are no explicit table locks in Sybase. Therefore, Share-None and Share-Read have no effect. Locking will be executed by the SQL engine.

## ***Views***

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted on Views.

Sybase allows updates on multiple table Views. If one of the segments of the DBPOS is updated, records shown on the screen may become inconsistent. It is important to note that Magic considers a View as a regular table. Magic should not be used to perform any rename or convert operations. If Magic is used to execute any rename or convert operations, Magic will display an error message and will not convert/rename the View in the database.

## ***Multiple Connections***

In a single Magic session, multiple databases can be used even though they may be located on different servers. The Sybase gateway opens one connection for each new server-user pair you are connected to. Specifically, connections to the same server but with different user names result in multiple connections.

The Sybase gateway opens an extra connection for each database server that uses Sybase stored procedures with Magic embedded SQL. The maximum number of connections appears in the beginning of the trace file.

When Magic executes a Direct SQL Command (DSQL) which has output parameters, Sybase will execute it on a dedicated connection. Therefore, if there is a parent task that executes a Direct SQL Command with output parameters, and, in record suffix it calls to a subtask , which executes a similar Direct SQL Command to the same server, then the gateway will look for an unused connection. Since the parent task is using the connection, the gateway will open a new connection. The above condition is true only when the user nests Direct SQL Command which has output parameters.

# **ODBC**

The following functions are available for the ODBC database.

## ***Locking***

No explicit table locks exist in ODBC. Therefore, Share-None and Share-Read have no effect. Locking is executed by the SQL engine.

## ***Tested ODBC Drivers***

The MagicGate Database Gateway for ODBC should work properly with all ODBC drivers. However, Magic is an Applications Generator, and the demands that Magic makes on an ODBC driver may be greater than those made by a regular program.

For this reason, Magic undertakes extensive tests of ODBC drivers, and prepares a list of tested drivers. See the Read Me file supplied with the gateway software for an up-to-date list of tested drivers.

## ***TroubleShooting***

Use the following layered approach to solve problems you may encounter when using the MagicGate Database Gateway for ODBC.

Test the ODBC Driver. If you can access the data successfully but the gateway is still not functioning correctly, test the ODBC driver itself using a tool such as Microsoft Query. If the data can be accessed through MSQuery, then the ODBC driver is correctly installed.

## ***Views***

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted on Views.

It is important to note that Magic considers a View as a regular table. Magic should not be used to perform any rename or convert operations. If Magic is used to execute any rename or convert operations, Magic will display an error message and will not convert/rename the View in the database.

## ***Check Driver Program***

A Check Driver program is included in the ODBC driver. Running this program tests the ODBC data source and prints a list of functions that are supported by the driver. The test will tell you whether or not the driver can be used with the MagicGate for ODBC.

If you cannot identify and correct the problem, then contact Magic Technical Support with information about the driver you are using, including the name of the developer and version of the product.

## ***Default Values***

ODBC does not support default values.

## ***Known Problems***

- Microsoft Access files with many fields, approximately above 227, causes an Access “Query too complex” error.
- When using a numeric field with the picture 2, when defining a file or table in Magic, change the Stored As parameter in the Column Properties dialog from the default Signed to Unsigned.
- It is illegal to use keywords from the data source or from the ODBC for column and table names. It is possible to view all the keywords in the Magic ODBC Gateway Log.

- No support exists for sort / temporary / application files in ODBC (data source).
- Restrictions and sizes are subject to limitations of the underlying database.

Please refer to the Read Me files supplied with your gateway software for the latest information on the operation of the MagicGate Database Gateway for ODBC.

## ***The ODBC Check Driver Utility: MGCHKDRV.EXE***

The ODBC Check Driver utility is included with the ODBC driver. Running this program tests the ODBC data source and prints a list of functions that are produced by the driver. The test will tell you whether or not the driver can be used with the MagicGate Database Gateway for ODBC.

The MGCHKDRV utility was built to check any driver of ODBC for information.

**The log file used in this example was produced by running the utility against an MS Access data source**, but the structure of the log is the same for all data sources and differences appear only in the retrieved text. Text appearing in courier 10 font comes from the log file.

Follow these steps to execute an SQL statement using the ODBC utility. The first part of each step, shown in *italics*, indicates the option that you can activate from the ODBC Test utility menu.

- 1 *Connect\Full Connect* – connect to the database
- 2 *Statement\AllocStmt* – allocate a statement handle
- 3 *Statement\SQLExecDirect* - execute the SQL statement you want
- 4 *Results\SQLGetData All* – get all the retrieved data
- 5 *Statement\FreeStmt* – free the statement
- 6 *Connect\Full Disconnect* – disconnect from the database

The Check Driver utility retrieves information about the data source in eight sections:

---

<b>Section No.</b>	<b>Information in Section</b>
1	Driver and DBMS Product Information
2	Data Source Information
3	SQL statements supported by the datasource
4	SQL Limits
5	DBMS Type support
6	SQL_KEYWORDS
7	Functions
8	Example for the syntax of DML commands

---

## ***ODBC Gateway - Data Source Information***

The following information appears in the mgchckdrv.log file, or in any other log file you specify.

### ***Section 1: Driver and DBMS Product Information***

The utility connects to the data source selected and sends the `SQLGetInfo` function, which returns general information about the driver associated with the connection's allocated handle.

In order to retrieve the same information, simply connect to the data source using `Connect\Full Connect`, and select `Connect\SQLGetInfo` with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

```
SQL_DBMS_NAME = ACCESS
```

A character string with the name of the DBMS product accessed by the driver.

```
SQL_ODBC_VER = 03.51
```

A character string with the version of ODBC to which the Driver Manager conforms. The version is of the form `##.##`, where the first two digits are the major version and the next two digits are the minor version. This is implemented solely in the Driver Manager.

```
SQL_DRIVER_NAME = odbcjt32.dll
```

A character string with the filename of the driver used to access the data source.

```
SQL_DRIVER_ODBC_VER = 03.51, major = 3, minor = 51
```

A character string with the version of ODBC that the driver supports. The version is of the form `##.##`, where the first two digits are the major version and the next two digits are the minor version. `SQL_SPEC_MAJOR` and `SQL_SPEC_MINOR` define the major and minor version numbers. For the version of ODBC described in this manual, these are 2 and 0, and the driver should return "02.00". If a driver supports `SQLGetInfo` but does not support this value of the `fInfoType` argument, the Driver Manager returns "01.00".

```
SQL_DRIVER_VER = 04.00.3513
```

A character string with the version of the driver and, optionally a description of the driver. At a minimum, the version is of the form `##.##.####`, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version.

```
SQL_DBMS_VER = 01.00.0000
```

A character string indicating the version of the DBMS product accessed by the driver. The version is of the form `##.##.####`, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The driver must render the DBMS product version in this form, but can also append the DBMS product-specific version as well. For example, "04.01.0000 Rdb 4.1".

```
SQL_DATABASE_NAME =  
C:\gateways\ODBCSDK\SMPLDATA\ACCESS\SAMPLE
```

A character string with the name of the current database in use, if the data source defines a named object called “database.”

**Note:** In ODBC 2.0, this value of `flInfoType` has been replaced by the `SQL_CURRENT_QUALIFIER` connection option. ODBC 2.0 drivers should continue to support the `SQL_DATABASE_NAME` information type, and ODBC 2.0 applications should only use it with ODBC 1.0 drivers.

```
SQL_ACTIVE_CONNECTIONS = 64
```

A 16-bit integer value specifying the maximum number of active connection handles that the driver can support. This value can reflect a limitation imposed by either the driver or the data source. If there is no specified limit or the limit is unknown, this value is set to zero.

```
SQL_ACTIVE_STATEMENTS = 0
```

A 16-bit integer value specifying the maximum number of active statement handles that the driver can support for a connection handle. This value can reflect a limitation imposed by either the driver or the data source. If there is no specified limit or the limit is unknown, this value is set to zero.

```
SQL_ODBC_API_CONFORMANCE = Level 1 Supported
```

A 16-bit integer value indicating the level of ODBC conformance:

- `SQL_OAC_NONE` = None
- `SQL_OAC_LEVEL1` = Level 1 supported
- `SQL_OAC_LEVEL2` = Level 2 supported

```
SQL_SEARCH_PATTERN_ESCAPE = \
```

A character string specifying what the driver supports as an escape character that permits the use of the pattern match meta-characters underscore (`_`) and percent (`%`) as valid characters in search patterns. This escape character applies only for those catalog function arguments that support search strings. If this string is empty, the driver does not support a search-pattern escape character. This `flInfoType` is limited to catalog functions.



```
SQL_SERVER_NAME = ACCESS
```

A character string with the actual data source-specific server name; useful when a data source name is used during `SQLConnect`, `SQLDriverConnect`, and `SQLBrowseConnect`.

## **Section 2: Data Source Information**

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo`, which returns general information about the data source associated with the connection's allocated handle.

In order to retrieve the same information, simply connect to the data source using `Connect\Full Connect`, and select `Connect\SQLGetInfo` with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

```
SQL_DATA_SOURCE_NAME = try-access
```

A character string with the data source name used during connection. If the application called `SQLConnect`, this is the value of the `szDSN` argument. If the application called `SQLDriverConnect` or `SQLBrowseConnect`, this is the value of the `DSN` keyword in the connection string passed to the driver. If the connection string did not contain the `DSN` keyword (such as when it contains the `DRIVER` keyword), this is an empty string.

```
SQL_ACCESSIBLE_TABLES = Y
```

A character string:

- "Y" if the user is guaranteed `SELECT` privileges to all tables returned by `SQLTables`,
- "N" if there may be tables returned that the user couldn't access.

```
SQL_CONCAT_NULL_BEHAVIOR =
```

Concatenation of a `NULL` value with no `NULL` value result is concatenation of non `NULL` valued

A 16-bit integer value indicating how the data source handles the concatenation of NULL valued character data type columns with non-NULL valued character data type columns:

- `SQL_CB_NULL` = Result is NULL valued.
- `SQL_CB_NON_NULL` = Result is concatenation of non-NULL valued column or columns.

```
SQL_DATA_SOURCE_READ_ONLY = N
```

A character string:

- "Y" if the data source is set to READ ONLY mode.
- "N" if the data source is not set to READ ONLY mode.

This characteristic pertains only to the data source itself; it is not a characteristic of the driver that enables access to the data source.

```
SQL_CURSOR_COMMIT_BEHAVIOR = Close cursors
```

A 16-bit integer value indicating how a COMMIT operation affects cursors and prepared statements in the data source:

- `SQL_CB_DELETE` = Close cursors and delete prepared statements. To use the cursor again, the application must re-prepare and re-execute the statement handle.
- `SQL_CB_CLOSE` = Close cursors. For prepared statements, the application can call `SQLExecute` on the statement handle without calling `SQLPrepare` again.
- `SQL_CB_PRESERVE` = Preserve cursors in the same position as before the COMMIT operation. The application can continue to fetch data or it can close the cursor and re-execute the statement handle without re-preparing it.

```
SQL_CURSOR_ROLLBACK_BEHAVIOR = Close cursors
```

A 16-bit integer value indicating how a ROLLBACK operation affects cursors and prepared statements in the data source:

- `SQL_CB_DELETE` = Close cursors and delete prepared statements. To use the cursor again, the application must again prepare and reexecute the statement handle.
- `SQL_CB_CLOSE` = Close cursors. For prepared statements, the application can call `SQLExecute` on the statement handle without calling `SQLPrepare` again.
- `SQL_CB_PRESERVE` = Preserve cursors in the same position as before the `ROLLBACK` operation. The application can continue to fetch data or it can close the cursor and re-execute the statement handle without re-preparing it.

`SQL_DEFAULT_TXN_ISOLATION` = `SQL_TXN_READ_COMMITTED`

A 32-bit integer that indicates the default transaction isolation level supported by the driver or data source, or zero if the data source does not support transactions. The following terms are used to define transaction isolation levels:

- **Dirty Read** - Transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 will have read a row that is considered to have never existed.
- **Non-repeatable Read** - Transaction 1 reads a row. Transaction 2 updates or deletes that row and commits this change. If transaction 1 attempts to reread the row, it will receive different row values or discover that the row has been deleted.
- **Phantom** - Transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 re-executes the statement that read the rows, it receives a different set of rows.

If the data source supports transactions, the driver returns one of the following bit-masks:

- `SQL_TXN_READ_UNCOMMITTED` = Dirty reads, non-repeatable reads, and phantoms are possible.

- SQL\_TXN\_READ\_COMMITTED = Dirty reads are not possible. Non-repeatable reads and phantoms are possible.
- SQL\_TXN\_REPEATABLE\_READ = Dirty reads and non-repeatable reads are not possible. Phantoms are possible.
- SQL\_TXN\_SERIALIZABLE = Transactions are serializable. Dirty reads, non-repeatable reads, and phantoms are not possible.
- SQL\_TXN\_VERSIONING = Transactions are serializable, but higher concurrency is possible than with SQL\_TXN\_SERIALIZABLE. Dirty reads are not possible. Typically, SQL\_TXN\_SERIALIZABLE is implemented by using locking protocols that reduce concurrency and SQL\_TXN\_VERSIONING is implemented by using a non-locking protocol such as record versioning. Oracle's Read Consistency isolation level is an example of SQL\_TXN\_VERSIONING.

SQL\_MULT\_RESULT\_SETS = N

A character string:

- "Y" if the data source supports multiple result sets.
- "N" if it does not.

SQL\_MULTIPLE\_ACTIVE\_TXN = Y

A character string:

- "Y" if active transactions on multiple connections are allowed
- "N" if only one connection at a time can have an active transaction.

SQL\_NEED\_LONG\_DATA\_LEN = N

A character string:

- "Y" if the data source needs the length of a long data value (the data type is SQL\_LONGVARCHAR,

SQL\_LONGVARBINARY, or a long, data source-specific data type) before that value is sent to the data source.

- "N" if it does not.

For more information, see `SQLBindParameter` and `SQLSetPos`.

```
SQL_OWNER_TERM = NULL STRING
```

A character string with the data source vendor's name for an owner. For example, `owner`, `Authorization ID`, or `Schema`.

```
SQL_NULL_COLLATION = NULLs are sorted at the low end of the list
```

A 16-bit integer value specifying where NULLs are sorted in a list:

- `SQL_NC_END` = NULLs are sorted at the end of the list, regardless of the sort order.
- `SQL_NC_HIGH` = NULLs are sorted at the high end of the list.
- `SQL_NC_LOW` = NULLs are sorted at the low end of the list.
- `SQL_NC_START` = NULLs are sorted at the start of the list, regardless of the sort order.

For retrieving the value of `SQL_AUTOCOMMIT`, described immediately below, the utility uses the existing connection to the data source selected and sends the function `SQLGetConnectOption`, which returns the current settings of a connection option.

In order to retrieve the same information, simply use `Connect\Full Connect` to connect to the data source, and select `Connect\SQLGetConnectOption` with the specific `fOption` you want (`SQL_AUTOCOMMIT` in this case). The result is in the `pvParam` field in the result window.

```
SQL_AUTOCOMMIT on connection = SQL_AUTOCOMMIT_ON
```

A 32-bit integer value that specifies whether to use auto-commit or manual-commit mode:

- `SQL_AUTOCOMMIT_OFF` = The driver uses manual-commit mode, and the application must explicitly commit or roll back transactions with `SQLTransact`.
- `SQL_AUTOCOMMIT_ON` = The driver uses auto-commit mode. Each statement is committed immediately after it is executed. This is the default. Note that changing from manual-commit mode to auto-commit mode commits any open transactions on the connection.

Important: Some data sources delete the access plans and close the cursors for all statement handles on a connection handle each time a statement is committed. Autocommit mode can cause this to happen after each statement is executed. For more information, see the `SQL_CURSOR_COMMIT_BEHAVIOR` and `SQL_CURSOR_ROLLBACK_BEHAVIOR` information types in `SQLGetInfo`.

`SQL_USER_NAME` = admin

A character string with the name used in a particular database, which can be different than login name.

### ***Section 3: SQL Statements Supported by the Datasource***

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo`, which returns information about the attributes of SQL statements supported by the data source associated with the connection's handle allocated.

In order to retrieve the same information, simply use *Connect\Full Connect* to connect to the data source, and select *Connect\SQLGetInfo* with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

`SQL_CORRELATION_NAME` = Correlation names are supported and can be any valid name.

A 16-bit integer indicating if table correlation names are supported:

- `SQL_CN_NONE` = Correlation names are not supported.

- `SQL_CN_DIFFERENT` = Correlation names are supported, but must differ from the names of the tables they represent.
- `SQL_CN_ANY` = Correlation names are supported and can be any valid user-defined name.

`SQL_IDENTIFIER_CASE` = Case sensitive, stored in mixed case

A 16-bit integer value as follows:

- `SQL_IC_UPPER` = Identifiers in SQL are case insensitive and are stored in upper case in system catalog.
- `SQL_IC_LOWER` = Identifiers in SQL are case insensitive and are stored in lower case in system catalog.
- `SQL_IC_SENSITIVE` = Identifiers in SQL are case sensitive and are stored in mixed case in system catalog.
- `SQL_IC_MIXED` = Identifiers in SQL are case insensitive and are stored in mixed case in system catalog.

`SQL_NON_NULLABLE_COLUMNS` = All columns must be nullable

A 16-bit integer specifying whether the data source supports non-nullable columns:

- `SQL_NNC_NULL` = All columns must be nullable.
- `SQL_NNC_NON_NULL` = Columns may be non-nullable (the data source supports the NOT NULL column constraint in CREATE TABLE statements).

`SQL_ODBC_SQL_CONFORMANCE` = Minimum grammar supported

A 16-bit integer value indicating SQL grammar supported by the driver:

- `SQL_OSC_MINIMUM` = Minimum grammar supported
- `SQL_OSC_CORE` = Core grammar supported
- `SQL_OSC_EXTENDED` = Extended grammar supported

SQL\_QUOTED\_IDENTIFIER\_CASE = Case insensitive, stored in mixed case

A 16-bit integer value as follows:

- SQL\_IC\_UPPER = Quoted identifiers in SQL are case insensitive and are stored in upper case in system catalog.
- SQL\_IC\_LOWER = Quoted identifiers in SQL are case insensitive and are stored in lower case in system catalog.
- SQL\_IC\_SENSITIVE = Quoted identifiers in SQL are case sensitive and are stored in mixed case in system catalog.
- SQL\_IC\_MIXED = Quoted identifiers in SQL are case insensitive and are stored in mixed case in system catalog.

### **Section 4: SQL Limits**

The utility uses the existing connection to the data source selected and sends the function `SQLGetInfo`, which returns information about the limits applied to identifiers and clauses in SQL statements, such as the maximum lengths of identifiers and the maximum number of columns in a select list.

**Important note:** Either the driver or the data source may impose limitations.

In order to retrieve the same information, simply connect to the data source with `Connect\Full Connect`, and select `Connect\SQLGetInfo` with the specific `fInfoType` you want. The result is in the `rgbInfoValue` field in the result window.

```
SQL_MAX_COLUMN_NAME_LEN = 64
```

A 16-bit integer value specifying the maximum length of a column name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

```
SQL_MAX_QUALIFIER_NAME_LEN = 260
```

A 16-bit integer value specifying the maximum length of a qualifier name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.



`SQL_MAX_TABLE_NAME_LEN = 64`

A 16-bit integer value specifying the maximum length of a table name in the data source. If there is no maximum length or the length is unknown, this value is set to zero.

`SQL_MAX_INDEX_SIZE = 255`

A 32-bit integer value specifying the maximum number of bytes allowed in the combined fields of an index. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_MAX_COLUMNS_IN_INDEX = 10`

A 16-bit integer value specifying the maximum number of columns allowed in an index. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_MAX_COLUMNS_IN_ORDER_BY = 10`

A 16-bit integer value specifying the maximum number of columns allowed in an ORDER BY clause. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_MAX_COLUMNS_IN_SELECT = 255`

A 16-bit integer value specifying the maximum number of columns allowed in a select list. If there is no specified limit or the limit is unknown, this value is set to zero.

`SQL_MAX_COLUMNS_IN_TABLE = 255`

A 16-bit integer value specifying the maximum number of columns allowed in a table. If there is no specified limit or the limit is unknown, this value is set to zero.

### ***Section 5: DBMS Type Support***

The utility uses the existing connection to the data source selected and sends the function SQLGetInfo with the flInfoType as SQL\_ALL\_TYPES, which returns information about all the data types supported. After receiving this information, the utility sends the function SQLGetTypeInfo for every data

type in order to see if the data type is searchable (in other words, if the data type can be used in a WHERE clause).

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Catalogue\SQLGetTypeInfo* with *fSQLType* = 'SQL\_ALL\_TYPES'. After that, select *Results\GetData All*. You will see the list of data types in the result window. To retrieve the SEARCHABLE parameter for each data type, select *Catalogue\SQLGetTypeInfo* with *fSQLOption* of the data type you want. Then select *Results\SQLBindCol* when *icol* = 9 (the SEARCHABLE parameter is the ninth parameter in this function). After that, select *Results\SQLFetch*. The last parameter in the result window (*rgbValue*) is the SEARCHABLE parameter value. The values are:

- 0 – SQL\_UNSEARCHABLE if the data type cannot be used in a WHERE clause.
- 1 – SQL\_LIKE\_ONLY if the data type can be used in a WHERE clause only with the LIKE predicate.
- 2 – SQL\_ALL\_EXCEPT\_LIKE if the data type can be used in a WHERE clause with all comparison operators except LIKE.
- 3 – SQL\_SEARCHABLE if the data type can be used in a WHERE clause with any comparison operator

The information in the utility log file is listed in 3 columns:

1. DBMS Type Name – The name of the data type in the DBMS.
2. SQL Data Type – the equivalent core SQL data type defined by ODBC
3. Searchable – a True\False value that indicates if the data type can appear in a WHERE clause.

DBMS TYPE NAME	SQL DATA TYPE	SEARCHABLE
GUID	INVALID SQLTYPE	FALSE
BIT	SQL_BIT	TRUE
BYTE	SQL_TINYINT	TRUE
LONGBINARY	SQL_LONGVARBINARY	FALSE
VARBINARY	SQL_VARBINARY	FALSE
BINARY	SQL_BINARY	FALSE
LONGCHAR	SQL_LONGVARCHAR	FALSE
CHAR	SQL_CHAR	TRUE
CURRENCY	SQL_NUMERIC	TRUE
INTEGER	SQL_INTEGER	TRUE
COUNTER	SQL_INTEGER	TRUE
SMALLINT	SQL_SMALLINT	TRUE
REAL	SQL_REAL	TRUE
DOUBLE	SQL_DOUBLE	TRUE
DATETIME	SQL_TIMESTAMP	TRUE
VARCHAR	SQL_VARCHAR	TRUE

### **Section 6: SQL Keywords**

The utility uses the existing connection to the data source selected and sends the function SQLGetInfo, with ‘SQL\_KEYWORDS’ in the flInfoType field.

This will return a character string containing a comma-separated list of all data source-specific keywords. This list does not contain keywords specific to

ODBC or keywords used by both the data source and ODBC. Keywords are reserved words – you can use them only in specific places according to the syntax.

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Connect\SQLGetInfo* with `SQL_KEYWORDS` in the `fInfoType` field. The result is in the `rgbInfoValue` field in the result window.

```
SQL_KEYWORDS =
ALPHANUMERIC, AUTOINCREMENT, BINARY, BYTE, COUNTER, CURRENCY,
DATABASE, DATABASENAME, DATETIME, DISALLOW, DISTINCTROW,
DOUBLEFLOAT, FLOAT4, FLOAT8, GENERAL, IEEEDOUBLE, IEEE SINGLE,
IGNORE, INT, INTEGER1, INTEGER2, INTEGER4, LEVEL, LOGICAL,
LOGICAL1, LONG, LONGBINARY, LONGCHAR, LONGTEXT, MEMO, MONEY,
NOTE, NUMBER, OLEOBJECT, OPTION, OWNERACCESS, PARAMETERS, PERCENT,
PIVOT, SHORT, SINGLE, SINGLEFLOAT, SMALLINT, STDEV, STDEVP,
STRING, TABLEID, TEXT, TOP, TRANSFORM, UNSIGNEDBYTE, VALUES,
VAR, VARBINARY, VARP, YESNO
```

## **Section 7: Functions**

The utility uses the existing connection to the data source selected and sends the function `SQLGetFunction`, with `SQL_API_ALL_FUNCTIONS` in the `fFunction` field.

This will return information about whether a driver supports the list of ODBC functions. Those functions are implemented in the Driver Manager. They can also be implemented in drivers. If a driver implements `SQLGetFunctions`, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself.

In order to retrieve the same information, simply connect to the data source with *Connect\Full Connect*, and select *Misc\SQLGetFunctions All*. The result in the result window will show a list of ODBC function names with a True/False value that identifies if they exist in the data source.

The information in the utility log file is listed in 3 columns:

1. Function – The name of the ODBC function.

2. Supported by driver – Supported\Not Supported according to the True\False value returned from the SQLGetFunction results.

3. Used By MAGIC – Required\Not Required by Magic.

**Important note:** Functions that are required by Magic but are not supported by the data source may cause further problems when working with Magic and ODBC with this data source.

Function	Supported by Driver	Used by Magic
SQLAllocConnect	Supported	Required
Allocates memory for a connection handle within the environment identified by the handle.		
SQLAllocEnv	Supported	Required
Allocates memory for an environment handle and initializes the ODBC call level interface for use by an application. An application must call SQLAllocEnv prior to calling any other ODBC function.		
SQLAllocStmt	Supported	Required
Allocates memory for a statement handle and associates the statement handle with the connection specified by a connection handle. An application must call SQLAllocStmt prior to submitting SQL statements.		
SQLBindCol	Supported	Required
Assigns the storage and data type for a column in a result set, including: A storage buffer that will receive the contents of a column of data The length of the storage buffer A storage location that will receive the actual length of the column of data returned by the fetch operation Data type conversion		
SQLCancel	Supported	Required
Cancels the processing of a statement's handle.		
SQLColAttributes	Supported	Not Required

Function	Supported by Driver	Used by Magic
SQLConnect	Supported	Required
Returns descriptor information for a column in a result set. It cannot be used to return information about the bookmark column (column 0). Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.		
SQLDescribeCol	Supported	Required
Loads a driver and establishes a connection to a data source. The connection handle references storage of all information about the connection, including status, transaction state, and error information.		
SQLDisconnect	Supported	Required
Returns the result descriptor column name, type, precision, scale, and nullability for one column in the result set. It cannot be used to return information about the bookmark column (column 0).		
SQLError	Supported	Required
Closes the connection associated with a specific connection handle.		
SQLExecDirect	Supported	Required
Returns error or status information.		
SQLExecute	Supported	Required
Executes a preparable statement, using the current values of the parameter marker variables if any parameters exist in the statement. SQLExecDirect is the fastest way to submit an SQL statement for one-time execution.		
SQLFetch	Supported	Required
Executes a prepared statement, using the current values of the parameter marker variables if any parameter markers exist in the statement.		
SQLFreeConnect	Supported	Required
Fetches a row of data from a result set. The driver returns data for all columns that were bound to storage locations with SQLBindCol.		
Releases a connection handle and frees all memory associated with the handle.		

<b>Function</b>	<b>Supported by Driver</b>	<b>Used by Magic</b>
SQLFreeEnv	Supported	Required
Frees the environment handle and releases all memory associated with the environment handle.		
SQLFreeStmt	Supported	Required
Stops processing associated with a specific statement handle, closes any open cursors associated with the statement handle, discards pending results, and, optionally, frees all resources associated with the statement handle.		
SQLGetCursorName	Supported	Not Required
Returns the cursor name associated with a specified statement handle.		
SQLNumResultCols	Supported	Required
Returns the number of columns in a result set.		
SQLPrepare	Supported	Required
Prepares an SQL string for execution		
SQLRowCount	Supported	Required
Returns the number of rows affected by an UPDATE, INSERT, or DELETE statement or by a SQL_UPDATE, SQL_ADD, or SQL_DELETE operation in SQLSetPos.		
SQLSetCursorName	Supported	Not Required
Associates a cursor name with an active statement handle. If an application does not call SQLSetCursorName, the driver generates cursor names as needed for SQL statement processing.		
SQLSetParam	Supported	Not Required
In ODBC 2.0, the ODBC 1.0 function SQLSetParam has been replaced by SQLBindParameter. For more information, see SQLBindParameter.		
SQLTransact	Supported	Required
Requests a commit or rollback operation for all active operations on all handle statements associated with a connection. SQLTransact can also request that a commit or rollback operation be performed for all connections associated with the environment handle.		

Function	Supported by Driver	Used by Magic
SQLColumns	Supported	Required

Returns the list of column names in specified tables. The driver returns this information as a result set on the specified statement's handle.

SQLDriverConnect	Supported	Required
------------------	-----------	----------

SQLDriverConnect is an alternative to SQLConnect. It supports data sources that require more connection information than the three arguments in SQLConnect, dialog boxes to prompt the user for all connection information and data sources that are not defined in the ODBC.INI file or registry.

SQLDriverConnect provides the following connection options:

Establish a connection using a connection string that contains the data source name, one or more user IDs, one or more passwords, and other information required by the data source.

Establish a connection using a partial connection string or no additional information; in this case, the driver Manager and the driver can each prompt the user for connection information.

Establish a connection to a data source that is not defined in the ODBC.INI file or registry. If the application supplies a partial connection string, the driver can prompt the user for connection information.

Once a connection is established, SQLDriverConnect returns the completed connection string. The application can use this string for subsequent connection requests.

SQLGetConnectOption	Supported	Required
---------------------	-----------	----------

Returns the current setting of a connection option.

SQLGetData	Supported	Required
------------	-----------	----------

Returns result data for a single unbound column in the current row. The application must call SQLFetch, or SQLExtendedFetch and (optionally) SQLSetPos to position the cursor on a row of data before it calls SQLGetData. It is possible to use SQLBindCol for some columns and use SQLGetData for others within the same row. This function can be used to retrieve character or binary data values in parts from a column with a character, binary, or data source-specific data type (for example, data from SQL\_LONGVARBINARY or SQL\_LONGVARCHAR columns).

SQLGetFunctions	Supported	Required
-----------------	-----------	----------

Returns information about whether a driver supports a specific ODBC function. This function is implemented in the Driver Manager; it can also be implemented in drivers. If a driver implements SQLGetFunctions, the Driver Manager calls the function in the driver. Otherwise, it executes the function itself.



<b>Function</b>	<b>Supported by Driver</b>	<b>Used by Magic</b>
SQLGetInfo	Supported	Required
Returns general information about the driver and data source associated with a connection handle.		
SQLGetStmtOption	Supported	Required
Returns the current setting of a statement option.		
SQLGetTypeInfo	Supported	Required
Returns information about data types supported by the data source. The driver returns the information in the form of an SQL result set. Important: Applications must use the type names returned in the TYPE_NAME column in ALTER TABLE and CREATE TABLE statements. SQLGetTypeInfo may return more than one row with the same value in the DATA_TYPE column.		
SQLParamData	Supported	Not Required
Is used in conjunction with SQLPutData to supply parameter data at statement execution time.		
SQLPutData	Supported	Not Required
Allows an application to send data for a parameter or column to the driver at statement execution time. This function can be used to send character or binary data values in parts to a column with a character, binary, or data source-specific data type (for example, parameters of the SQL_LONGVARBINARY or SQL_LONGVARCHAR types).		
SQLSetConnectOption	Supported	Required
Sets options that govern aspects of connections.		
SQLSetStmtOption	Supported	Required
Sets options related to a statement handle. To set an option for all statements associated with a specific connection handle, an application can call SQLSetConnectOption.		
SQLSpecialColumns	Supported	Required
Retrieves the following information about columns within a specified table: <ul style="list-style-type: none"> <li>The optimal set of columns that uniquely identifies a row in the table.</li> <li>Columns that are automatically updated when any value in the row is updated by a transaction.</li> </ul>		

Function	Supported by Driver	Used by Magic
SQLStatistics	Supported	Required
Retrieves a list of statistics about a single table and the indexes associated with the table. The driver returns the information as a result set.		
SQLTables	Supported	Required
Returns the list of table names stored in a specific data source. The driver returns the information as a result set.		
SQLBrowseConnect	Not Supported	Not Required
Supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to SQLBrowseConnect returns successive levels of attributes and attribute values.		
When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by SQLBrowseConnect. A return code of SQL_SUCCESS or SQL_SUCCESS_WITH_INFO indicates that all connection information has been specified and the application is now connected to the data source.		
SQLColumnPrivileges	Not Supported	Not Required
Returns a list of columns and associated privileges for the specified table. The driver returns the information as a result set on the specified statement's handle.		
SQLDataSources	Supported	Not Required
Lists data source names. This function is implemented solely by the Driver Manager.		
SQLDescribeParam	Not Supported	Not Required
Returns the description of a parameter marker associated with a prepared SQL statement.		
SQLExtendedFetch	Supported	Required
Extends the functionality of SQLFetch in the following ways: It returns rowset data (one or more rows), in the form of an array, for each bound column. It scrolls through the result set according to the setting of a scroll-type argument.		
SQLExtendedFetch works in conjunction with SQLSetStmtOption. To fetch one row of data at a time in a forward direction, an application should call SQLFetch.		

Function	Supported by Driver	Used by Magic
SQLForeignKeys	Not Supported	Not Required
<p>SQLForeignKeys can return:</p> <ul style="list-style-type: none"> <li>A list of foreign keys in the specified table (columns in the specified table that refer to primary keys in other tables).</li> <li>A list of foreign keys in other tables that refer to the primary key in the specified table.</li> </ul> <p>The driver returns each list as a result set on the specified statement handle.</p>		
SQLMoreResults	Supported	Not Required
<p>Determines whether there are more results available on a statement handle containing SELECT, UPDATE, INSERT, or DELETE statements and, if so, initializes processing for those results.</p>		
SQLNativeSql	Supported	Not Required
<p>Returns the SQL string as translated by the driver.</p>		
SQLNumParams	Supported	Not Required
<p>Returns the number of parameters in an SQL statement.</p>		
SQLParamOptions	Supported	Not Required
<p>Allows an application to specify multiple values for the set of parameters assigned by SQLBindParameter. The ability to specify multiple values for a set of parameters is useful for bulk inserts and other work that requires the data source to process the same SQL statement multiple times with various parameter values. An application can, for example, specify three sets of values for the set of parameters associated with an INSERT statement, and then execute the INSERT statement once to perform the three insert operations.</p>		
SQLPrimaryKeys	Not Supported	Not Required
<p>Returns the column names that comprise the primary key for a table. The driver returns the information as a result set. This function does not support returning primary keys from multiple tables in a single call.</p>		
SQLProcedureColumns	Supported	Not Required
<p>Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures. The driver returns the information as a result set on the specified statement handle.</p>		

Function	Supported by Driver	Used by Magic
SQLProcedures	Supported	Not Required
Returns the list of procedure names stored in a specific data source. Procedure is a generic term used to describe an executable object, or a named entity that can be invoked using input and output parameters, and which can return result sets similar to the results returned by SQL SELECT expressions.		
SQLSetpos	Supported	Not Required
Sets the cursor position in a rowset and allows an application to refresh, update, delete, or add data to the rowset.		
SQLSetScrollOptions	Supported	Not Required
Sets options that control the behavior of cursors associated with a statement handle. SQLSetScrollOptions allows the application to specify the type of cursor behavior desired in three areas: concurrency control, sensitivity to changes made by other transactions, and rowset size. <b>Note:</b> In ODBC 2.0, SQLSetScrollOptions has been superceded by the SQL_CURSOR_TYPE, SQL_CONCURRENCY, SQL_KEYSET_SIZE, and SQL_ROWSET_SIZE statement options. ODBC 2.0 drivers must support this function for backwards compatibility; ODBC 2.0 applications should only call this function in ODBC 1.0 drivers. If an application calls SQLSetScrollOptions, a driver must be able to return the values of the aforementioned statement options with SQLGetStmtOption. For more information, see SQLSetStmtOption.		
SQLTablePrivileges	Not Supported	Not Required
Returns a list of tables and the privileges associated with each table. The driver returns the information as a result set on the specified statement handle.		
SQLDrivers	Supported	Not Required
Lists driver descriptions and driver attribute keywords. This function is implemented solely by the Driver Manager.		
SQLBindParameter	Supported	Required
Binds a buffer to a parameter marker in an SQL statement. <b>Note:</b> This function replaces the ODBC 1.0 function SQLSetParam.		

## Section 8: Syntax of DML Commands Example

The utility uses the existing connection to the data source selected and sends the function `SQLExecDirect` with a SQL command in the `szSqlStr` field.

After each command, the log lists if the command was successful or not.

Note that the table and index are in fact actually created and dropped from the database that the data source points to.

```
CREATE TABLE T11025 (FLD1 CHAR(1), FLD2 CHAR(1))
**** Succeeded ****

CREATE UNIQUE INDEX I11025 ON T11025 (FLD1)
**** Succeeded ****

DROP INDEX I11025 ON T11025
**** Succeeded ****

DROP TABLE T11025
**** Succeeded ****
```

The utility disconnects from the data source using the `SQLDisconnect` function.

In order to do the same, simply select *Connect\Full Disconnect*.

```
*****      Disconnected Successfully ... *****
*****
```

## DB2

The following functions are available for the DB2 database.

## ***Timeout Locks***

A physical lock is a method that ensures that no one can modify a record. Specifically, from the moment a user locks the record until that user releases the record, the record cannot be modified by another user. The physical lock is implemented as follows: When Magic locks a record according to its locking strategy, Magic issues a Select statement with a For Update clause. The For Update clause prevents the application from making changes to the record until the end of the transaction.

**Note:** In an abnormal situation, if lock timeout detection is not used, the application must wait for a lock to be released. To avoid stalling your program in such a case, you can use the lock timeout configuration parameter to set the maximum time that your application will wait to obtain a lock. The following should be entered from the command line processor:

```
update database configuration for database alias using  
locktimeout xx
```

## ***Views***

A View must have a virtual unique index defined. Insert, Update, and Delete operations are permitted.

It is important to note that Magic considers a View as a regular table. Magic should not be used to perform any rename or convert operations. If Magic is used to execute any rename or convert operations, Magic will display an error message and will not convert/rename the View in the database.

## ***Using DB2 Handles***

The DB2 Gateway is written in CLI, which requires the use of handles. Handles are data objects that contain information about a single SQL statement. The handles are allocated at the beginning and released when the SQL statement is no longer used. Each Magic cursor is a SQL statement that correlates to a DB2 Handle. The statement handle is allocated the first time

the cursor is opened; specifically the first time Magic tries to fetch records from the Table/File.

The statement handle is released only when the cursor is released/dropped. This is done because the cursor can be opened more than once. Link cursors are reopened for each record fetched, while main cursors are not reopened every time you go to the next page. Reallocating the statement each time, will reduce performance.

As a result, for non-resident tasks the handle is released when the task is closed, or for resident tasks, the handle is released when you exit Magic runtime.

In DB2, the open handles limit is based on the isolation level: for each isolation level there is a limit of open handles. In Magic, the isolation level can be set in the database level, and tables can be mapped to a different database in Magic. This will result in different handles remaining open in different isolation levels, and the number of open handles remaining higher.

## **Informix**

The following functions are available for the Informix database.

### ***Views and Fragmented Tables***

Tables consisting of Views, or fragmented tables without a ROWID must have any unique index which can be either virtual or real. Defining virtual indexes is explained in *The Magic Guide to Application Development*. Certain views defined by Informix as non-modifiable views are Read-Only and Insert. Update and Delete operations are not permitted.

It is important to note that Magic considers Views and fragmented tables as regular tables. Magic should not be used to perform any rename or convert operations. An attempt to use these operations will cause Magic to display an error message and will not convert/rename the View in the database.

A View that is defined on more than one table does not have a ROWID, and that is the reason that the Position parameter on the SQL tab in the Table Properties dialog should be set to Unique Index and not to the default.

### ***Locking***

Table Locking is available only within a Magic transaction. Magic will ignore the request if it is issued outside of a transaction. Therefore, Share None refers to Exclusive and Share Refer refers to Share Lock.

In Sybase, the user can control the physical locking strategy, via the parameter isolation level which can be specified in the DBMS level. This is carried out either through Magic or the DBS and file level, which uses the “database info” field.

The above condition is true only if the user is using Sybase Version 11.



## ***Multiple Connections***

Multiple Database Connections are not supported by the Informix5 gateway. Access to other Informix databases is possible through the first Informix database accessed. Therefore, all secondary databases must be homogenous with the first, in terms of Online or Standard engine, with or without logging, and whether or not the database is ANSI.

The default Informix7 Gateway setting supports Multiple Database Connections. Therefore, multiple non-homogenous databases may be accessed at the same time. When using multiple connections, the ability of the Informix Server to perform a Two-Phase Commit is lost.

For the Informix Server to perform a Two Phase Commit, you must define a *desired connecting database and server*, meaning the database and server that the connection will be directed to, in addition to the *accessed database and server*, meaning the database that holds the accessed tables. By default, the *desired connecting database and server* is equal to the *accessed database and server*. To use different databases or servers, the following two Database Information parameters should be defined:

```
SQLCVDATABASE="desired connecting database"  
SQLCVSERVER="desired connecting server"
```

If the *accessed database* and the *desired connecting database* reside on the same server, there is no need to define the SQLCVSERVER. In this case, the default values are sufficient. For all other cases, both parameters should be defined.

## ***Text and Byte Data Types***

The user must enter the appropriate picture, after performing a Get Definition on an Informix table, which contains a column of type TEXT or BYTE. By default, the field that receives the column will have a picture of 0. This picture is invalid, and the user must enter a valid picture between 1 and 32000.

## Compatibility with Previous Versions

The following flags will be inserted into special fields in the object properties, and will no longer be supported in the Database Information:

Flag	Property	Oracle	DB2	MS-SQL	Informix	Sybase	ODBC
DBMS	Maximum Connections			SQL_MAX_SERVER_CONNECTIONS		SQL_MAX_SERVER_CONNECTIONS	
DBMS	Show Plan		SQL_SHOWPLAN	SQL_SHOWPLAN		SQL_SHOWPLAN	
DBMS	Isolation Level		SQL_ISOLATION_LEVEL	SQL_ISOLATION_LEVEL		SQL_ISOLATION_LEVEL	
DB	Check Existence	SQL_TAB_EXIST					
DB	Hint			SQL_HINT		SQL_HINT	
DB	Array Size					SQL_ARRAY	
DB	Connect String	SQL_CONNECT					
Table	Check Existence	SQL_TAB_EXIST					
Table	Type	SQL_VIEW					
Table	Owner	SQL_OWNER	SQL_OWNER	SQL_OWNER	SQL_OWNER	SQL_OWNER	SQL_OWNER

Flag	Property	Oracle	DB2	MS-SQL	Informix	Sybase	ODBC
Table	Hint	SQL_HINT		SQL_HINT		SQL_HINT	
Table	Array Size					SQL_ARRAY	
Index	Clustered			Clustered		Clustered	Clustered
Index	Hint	SQL_HINT					
Index	Drop during Reindex		CONSTRAINT	CONSTRAINT		CONSTRAINT	
Index	Index		DPOS	DPOS		DPOS	DPOS
Index	Owner				SQL_OWNER		
Column	Type	SQL_TYPE	SQL_TYPE	SQL_TYPE	SQL_TYPE	SQL_TYPE	SQL_TYPE
Column	User Type		SQL_USERTYPE				

The following flags will be supported in the Database Information fields:

- Informix
  - Database Parameters  
SQLCVDATABASE  
SQLCVSERVER
- MS-SQL
  - Database Parameters  
MSDATE changed to SQL\_DATETOALPHA
- Oracle
  - Database Parameters  
ALPHA\_DATE changed to SQL\_DATETOALPHA  
NO\_DEFER
  - Table Parameters  
CLUSTER  
INITRANS  
MAXTRANS  
PCTFREE  
PCTUSED  
STORAGE  
TABLESPACE
- Sybase
  - Database Parameters  
SYBDATETODATE changed to SQL\_DATETOALPHA
- DBL
  - Database Parameters  
SQL\_ISOLATION\_LEVEL

## *Properties Supported by SQL Gateways*

Repository	Property	Oracle	DB2	MS-SQL	Informix	Sybase	ODBC
DBMS	Log Level	x	x	x	x	x	x
DBMS	Log Name	x	x	x	x	x	x
DBMS	Log Sync	x	x	x	x	x	x
DBMS	Check Existence	x	x	x	x	x	x
DBMS	Maximum Connections			x		x	
DBMS	Show Plan			x		x	
DBMS	Isolation Level		x	x	x	x	
DB	Check Existence	x	x	x	x	x	x
DB	Hint			x		x	
DB	Array Size		x	x		x	
DB	Connect String	x					
Table	Check Existence	x	x	x	x	x	x
Table	Type	x	x	x	x	x	x
Table	Position	x	x	x	x	x	x
Table	Index	x	x	x	x	x	x
Table	Owner	x	x	x	x	x	x
Table	Hint	x		x		x	
Table	Cursor			x			

Repository	Property	Oracle	DB2	MS-SQL	Informix	Sybase	ODBC
Table	Array Size	x	x	x		x	
Index	Clustered			x	x	x	
Index	Hint	x		x		x	
Index	Drop during Reindex	x	x	x	x	x	x
Index	Owner				x		
Column	Type	x	x	x	x	x	x
Column	User Type		x	x		x	

# Basic SQL Command Syntax



---

STATEMENT	<b>SELECT</b>
DESCRIPTION	Retrieves data from a table
SYNTAX	<b>SELECT</b> <i>column_name ,function</i> <b>FROM</b> <i>table1 alias [ ,table2 alias,...]</i> <b>[WHERE</b> <i>condition</i> <b>]</b> <b>[GROUP BY</b> <i>column_name [,column_name,...]</i> <b>[HAVING</b> <i>condition</i> <b>[ORDER BY</b> <i>column1 [ASC / DESC] [ , column2</i> <b>[ASC / DESC],... ]</b>
STATEMENT	<b>INSERT</b>
DESCRIPTION	Adds new rows of data to the database
SYNTAX	<b>INSERT INTO</b> <i>table_name [(column1,column2,...)]</i> <b>VALUES</b> ( <i>value1,value2,...</i> ) <b>[SELECT</b> <i>statement</i> <b>]</b>

STATEMENT        **DELETE**  
DESCRIPTION      Removes rows of data from the database  
SYNTAX            **DELETE FROM** *table name*  
                      [**WHERE** *condition*]

STATEMENT        **UPDATE**  
DESCRIPTION      Modifies existing database data  
SYNTAX            **UPDATE** *table\_name*  
                      **SET** *column1 = value1, column2 = value2...*  
                      [**WHERE** *condition*]

STATEMENT        **CREATE TABLE**  
DESCRIPTION      Adds a new table to the database  
SYNTAX            **CREATE TABLE** *table\_name* (*column1 data\_type,*  
                      *column2 data\_type,...*)

STATEMENT        **ALTER TABLE**  
DESCRIPTION      Changes the structure of an existing table  
SYNTAX            **ALTER TABLE** *table\_name*        **ADD**  
*column\_name column\_definition*  
                      **DELETE**  
                      **MODIFY**



STATEMENT            **CREATE VIEW**  
DESCRIPTION        Adds a new view to the database  
SYNTAX              **CREATE VIEW** *view\_name* [(*column definition*)]  
                         **AS SELECT** *sql\_statement*

STATEMENT            **CREATE INDEX**  
DESCRIPTION        Builds an index for a column  
SYNTAX              **CREATE [UNIQUE] INDEX** *index\_name* **ON**  
*table\_name* (*column\_name*  
                         [DESC],...)

[This page intentionally left blank]

# *Example Database Information*

---

# B

## *Products*

Pid	Pname
1	bolt
2	nut
3	screw
4	nail

## *Suppliers*

Sid	Sname	Address
10	Acme	Boston
20	Brown	Dallas
30	Central	Chicago

## *Inventory*

Sid	Pid	Amount	Date
10	1	10	
10	2	20	
20	2	16	
30	3	20	

## *Offices*

City	Salespeople	Target	Sales	Region
Chicago	3	750,000	750,042	Eastern
Los Angeles	4	800,000	835,915	Western
New York	4	700,000	692,637	Eastern
Atlanta	2	350,000	367,911	Eastern
Denver	1	200,000	186,042	Western

## *Departments*

Deptno	Dname	Location
10	Accounting	Los Angeles
20	Marketing	San Francisco
30	Operations	Norfolk
40	Sales	New York
50	Research	Berkeley

# Employees

---

Emp No	Ename	Job	Mgr	Hiredate	Sal	Comm	Age	Deptno
3567	Allen	Sales	3970	15-Apr-85	4300	2000	33	40
3891	George	Sales	3970	01-Jan-94	4400	3000	32	40
3092	Turnbull	Sales	3970	15-Jun-93	4050	5050	40	40
3667	Markson	Clerk	3373	17-Oct-90	3000		27	30
3559	Jordan	Analyst	3408	03-May-89	4250		30	50
3123	Walker	Pres.		13-Mar-8	5500		40	10
3472	Major	Mgr	3123	13-Mar-8	4750		43	10
3373	Klein	Analyst	3408	02-Feb-93	3900		41	50
3162	Barton	Mgr	3123	02-Dec-87	4200		35	20
3408	Eiden	Mgr	3123	04-Aug-89	4700		33	50
3582	Anton	Clerk	3162	22-Jan-92	3500		42	20
3115	Carlisle	Clerk	3162	15-Mar-93	3375		22	10
3012	Bonfiglio	Analyst	3408	27-Jul-91	4175		42	50
3970	O'Neil	Mgr	3123	15-May-94	4850		43	40

---

[This page intentionally left blank]

# Index

---

## A

---

- Advanced Select Statements 7-71 7-73 7-75  
7-77 7-79 7-81 7-83 7-85 7-87
  - Arithmetic Expressions 7-72
  - Null Functions 7-73
- Advanced SELECT Statements
  - Concatenation Function 7-74
  - Expressions 7-71
  - GROUP BY Clause 7-78
  - Group Functions 7-77
  - HAVING Clause 7-80
  - Multi-table Queries 7-75
  - Nested 7-82
- Assist utility 22-251
- Assist Utility
  - how to use 22-252
- Automatic Program Generator 25-278

## B

---

- Basic SQL Command Syntax A-327 A-329
- Batch Immediate Locking Strategy 21-242
- Batch Tasks 23-263
- BLOBS Locking 21-244

## C

---

- Client/Server 23-267
  - Client/Server Architecture 14-139 14-141  
14-143
    - RDBMS 14-142
  - Column Properties 17-175
    - Attribute 17-175
    - Database Default 17-176
    - Database Information 17-178
    - DB Column Name 17-177
    - Information 17-177
    - Name 17-175
    - Null Allowed 17-176
    - Null Default 17-176
    - Null Value 17-176
    - Stored As 17-177
    - Type 17-178
    - User Type 17-178
  - Columns
    - selecting 5-47
    - compatibility with previous versions 25-322
  - Connect String 3-39
  - Constants 2-33
    - numeric 2-33
  - Constraints
    - Database Support 9-99
    - Multiple Table 9-98
    - Single Table 9-97
  - Cursor Routines 13-136
- ## D
- 
- Data Conversion 7-84
  - Data Definition Language 1-26 4-42
  - Data Definition Rules 18-204
  - Data Dictionary
    - Oracle 11-110
  - Data Manipulation

- Magic 13-135
- Data Manipulation Language 1-26
- Data Replication 10-104
- Data Types
  - ANSI/ISO 2-29 2-31 2-33 2-35
  - DB2 18-202
  - differences 2-32
  - enforcing 18-193
  - extended 2-31
  - Informix 18-198
  - Limitations 2-32
  - Magic 18-193
  - ODBC 18-203
  - RDBMS 2-34
  - SQL Server 18-201
  - Sybase 18-199
- Data Types
  - Oracle 18-200
- Database
  - creating 4-41
  - querying 1-26
  - updating 1-27
- Database Administrator 1-27
- Database Application Development 1-25
- Database Connection
  - Informix 15-149
  - MS-SQL 15-150
  - Oracle 15-148
  - permission 15-150
  - Sybase 15-147
- Database Information 17-170
  - Array Size 17-173
  - Check Existence 17-171
  - Cursor 17-172
  - DB Table 17-173
  - Default Position 17-171

- Hint 17-172
- Index 17-171
- Owner 17-171
- Position 17-171
- Table Type 17-172
- Database Name 17-168
- Date and Time Mapping 18-195
- DB Table 17-168
- DB2
  - Timeout Locks 25-318
  - Views 25-318
- DB2
  - Using DB2 Handles 25-318
- DBMS Functions
  - Access Control 1-21
  - Data Definition 1-21
  - Data Integrity Control 1-21
  - Data Manipulation 1-21
  - Data Retrieval 1-21
  - Data Sharing 1-21
- Default Mapping
  - Magic 18-191
- Defining Date Columns 18-195
  - size 18-195
  - SQL type 18-196
  - stored as 18-195

## **E**

---

- Error Handling 22-258 25-280
  - DBERR Function 22-258
- Error Recovery Control Options 21-239
- Example Database Information B-331 B-333
- Explicit Embedded SQL Elements
  - create 22-250
- Explicit SELECT Statements
  - behavior 22-255



Explicit SQL 22-247 22-249 22-251 22-253  
22-255 22-257

Open Object Menu 22-248  
using 22-248

## **F**

---

### Flags

how to set 20-227  
Log Sync. (Yes/No) 20-228  
Show Plan (Yes/No) 20-228  
where to set 20-229

Flags for Database Information Fields 25-324

Informix 25-324  
MS-SQL 25-324  
Oracle 25-324  
Sybase 25-324

Flow Monitor 20-230

## **G**

---

### Gateway

Communication 14-140  
Definition SQL 14-141  
Full SQL 14-141  
Informix 13-133  
MS-SQL 13-133  
ODBC 13-133  
Oracle 13-133  
Sybase 13-133

### Gateways

Magic 13-132

Get Definition 17-183

Get Table Definition Mapping 18-192

## **I**

---

Incremental Locate 19-225

Index Definition 17-180

Name 17-181

Segments 17-181

Type 17-181

### Index Properties

Clustered 17-183

DB Index Name 17-182

Direction 17-181

Drop During Reindex 17-183

Index Type 17-182

Information 17-182

Range Mode 17-182

### Indexes

Non-unique 19-209

Unique 19-209

Informix 11-115

Locking 25-320

Multiple Connections 25-321

Text and Byte Data Types 25-321

Views and Fragmented Tables 25-320

Input Parameters 22-251

Intrans Function 21-245

ISAM-RDBMS 24-271

Adding Direct SQL Stored Procedures  
24-273

Convert 24-271

general settings 24-271

Indexes 24-272

Links 24-273

Magic Locking 24-273

select all columns with no default values  
24-272

select all position columns 24-272

Table Repository 24-271

Transactions 24-273

Isolation levels 10-106

Isolation Levels 21-244

## **J**

---

- Join Operation 19-212
  - behavior 19-214
  - locking 19-214
  - parameters 19-213
  - usage 19-214
- K** 

---
- Key Definition
  - usage 23-259
- L** 

---
- LCK Parameter 21-241
- Link Join Operation 19-212
  - delete 19-217
  - insert 19-216
  - update 19-217
  - usage considerations 19-216
- Link Operation
  - regular 19-211
- Locking 10-105
  - duration 10-108
  - enforcing 10-108
- Locking Example
  - DB2 19-215
  - Informix 19-215
  - Oracle 19-215
- Locking levels 10-107
- Locking Processing 21-231 21-233 21-235  
21-237 21-239 21-241 21-243 21-245
- Locking Strategies 21-233
  - After Modify 21-234
  - Before Update 21-234
  - Immediate 21-233
  - No Lock 21-235
  - On Modify 21-234
- Log File
  - how to work with 20-229

- Logical Locks 21-241 21-243
- Lowest Common Denominator 24-275

## **M** ---

- Magic 8
  - what's new 24-269
- Magic Gateway
  - Version Convention 16-155
- Magic Gateway Log 20-227
- Magic Profiler 20-230
- MAGIC\_DATABASES Settings 16-160
  - Array Size 16-164
  - Change Toolkit Tables 16-163
  - Check Existence 16-164
  - Connect String 16-162
  - Database Information 16-163
  - Database Server 16-162
  - Dbname 16-161
  - External Use in Magic 5.x & 6.x 16-164
  - Hint 16-163
  - Location 16-162
  - Magic Locking 16-163
  - Magic Server 16-161
  - Name 16-161
  - User Name and User Password 16-162
- Magic\_DBMS Flags 16-158
- MAGIC\_DBMS Flags
  - DBMS Parameters 16-159
  - DBMS Properties 16-159
  - Default Float Picture 16-158
- MAGIC\_ENV Flags
  - Display Full Messages 16-157
  - ISAM Transactions 16-156
  - Multi-user 16-157
- MagicGate Database
  - Naming Convention 16-154

MagicGate Database Gateway  
     configuring 16-153  
 MagicGate Database Gateways 13-136  
 Mapping  
     existing data 18-196  
 Mapping Data Types 18-191 18-193 18-195  
 18-197 18-199 18-201 18-203 18-205  
 Migrating from Magic 6/7 to Magic 8 24-270  
 Modifying Data  
     Delete 8-92  
     Insert 8-89  
     Update 8-93  
 MS-SQL  
     DB Commands 25-286  
     Locking 25-286  
     Multi-connections 25-286  
     Optimizer Hints 25-285  
     Relevant Parameters 25-287  
     Views 25-286  
 Multiple Connections 16-166  
 Multiple Databases 16-166  
 Multi-user Flag 21-232  
**O** \_\_\_\_\_  
 Object Privilege 11-116  
 ODBC 25-291  
     Check Driver Program 25-292  
     Check Driver Utility 25-293  
     Known Problems 25-292  
     Locking 25-291  
     Tested ODBC Drivers 25-291  
     Troubleshooting 25-291  
     Views 25-292  
 On Record Locked Parameter 21-240  
     Abort 21-240  
     Retry 21-240  
     Skip 21-240  
 One-Way Behavior 19-223  
 Online vs. Batch 22-255  
 Operating System  
     Oracle 11-114  
 Optimizer  
     hinting 12-121  
     MS-SQL 12-121  
     Sybase 12-121  
     Tracking 12-121  
 Optimizer Hints 21-244  
 Oracle 25-281  
     Create Table Parameters 25-283  
     Deferred Execution of SQL Steps 25-284  
     Load Definition from a Remote Server  
     25-283  
     Multi-connections 25-282  
     Optimizer Hints 25-281  
     Stored Procedure 25-281  
     Unique Views 25-282  
     Views 25-282  
 Output Parameters 22-251  
**P** \_\_\_\_\_  
 Physical Locks 21-241  
 Position Index 19-209  
 Programming with SQL 7-87  
**R** \_\_\_\_\_  
 Range Definition 23-262  
 Ranging  
     CNDRANGE Function 19-218  
 RDBMS Data Types  
     Magic equivalents 18-197  
 RDBMS Environment 15-145 15-147 15-149  
 15-151  
     DB2 15-146

- Informix 15-146
- MS-SQL 15-146
- Oracle 15-146
- Sybase 15-145
- RDBMS Tools 20-229
- RDBMS-ISAM 24-274
- RDBMS-RDBMS 24-274
- Recommendations 22-256
- Reconnect 3-39
- Record Level
  - No 21-238
  - ON Lock 21-237
  - Prefix 21-237
  - Suffix 21-238
  - Update 21-238
- Referential Integrity 9-98
- Repository 13-134
- Restrictions on Using Embedded SQL 22-257
- Result Database 25-278
- Result Database as Input Database 22-255
- Result Database Different from Input Database 22-256
- ROLLBACK Function 21-244
- Row Selection
  - WHERE Clause 6-57
- Rules 25-280
- S** \_\_\_\_\_
- Search Conditions 6-58
- Security 25-279
  - data 11-113
  - System 11-113
- Select Statements 25-277
- Share Mode 21-232
- Sort
  - behavior 19-226
  - usage 19-225
  - usage considerations 19-226
- Sort Using RDBMS 19-225
- Sort/Temporary Database 25-280
- Sorting 23-264
  - query results 5-53
- SQL
  - engine 1-20
  - Server 3-37
  - Structure 1-20
  - user interface 3-38
- SQL Command 22-250
- SQL Command Automatic Program Generator - APG 22-253
  - how to use 22-254
  - how to view 22-254
- SQL Command Object 25-277
- SQL Concepts
  - Column 1-24
  - Cursor 1-25
  - Null Value 1-24
  - Row 1-24
  - Table 1-24
  - View 1-25
- SQL DATETOALPHA Parameter 25-279
- SQL Gateways
  - properties supported 25-325
- SQL Reconnect 3-39
- SQL Where
  - clause 19-219
- SQL Where Behavior 19-221
- SQL Where Clause
  - Add to Where Clause 19-219
  - column list 19-219
  - considerations 19-222
  - Full Where Clause 19-220

- usage 19-219
- Stored Procedure 9-100 25-278
- Stored Procedures 23-266
  - executing 9-101
- Sybase
  - Locking 25-290
  - Multiple Connections 25-290
  - Optimizer Hints 25-289
  - Views 25-290
- System Privilege 11-115
- System tables 11-109
  - usage 11-111
- System-Stored Procedures
  - MS-SQL 11-112
  - Sybase 11-112

**T** \_\_\_\_\_

- Table Access Mode 21-232
- Table Modification 17-188
- Table Properties 17-174
  - Resident & Cache Strategy 17-170
- Table Repository 17-167
- Technical Information 25-277 25-279 25-281 25-283 25-285 25-287 25-289 25-291 25-293 25-295 25-297 25-299 25-301 25-303 25-305 25-307 25-309 25-311 25-313 25-315 25-317 25-319 25-321 25-323 25-325
- Terminology
  - Magic 13-131
  - SQL 13-131
- TIME Attribute 18-197
- Trace facility
  - Oracle 12-122
- Trace Facility
  - SQL 12-122
- Transaction 10-103 10-105 10-107

- Transaction Error Parameter 21-240
- Transaction Level Rules 21-239
- Transaction Levels 21-236
  - Record Level 21-236
  - Task Level 21-236
- Transaction Processing 13-136 21-231 21-233 21-235 21-237 21-239 21-241 21-243 21-245
- Transactions 23-263
  - Online Tasks 23-263
- Triggers 9-102 23-266 25-280
- Two Phase Commit 25-282
- Two-Phase COMMIT 10-104
- Two-Way Behavior 19-223

**U** \_\_\_\_\_

- Unique Identifier 19-207
  - Magic 19-208
  - RDBMS 19-207

**V** \_\_\_\_\_

- View Definition Loading 17-188
- Views 4-45 23-265

**W** \_\_\_\_\_

- Windows Icons
  - MS-SQL 15-151
  - Oracle 15-151
  - Sybase 15-151

4182-0400-0001