



User's Guide

**Borland
SQL Link**

Version 1.0

Borland International, Inc. 1800 Green Hills Road
P.O. Box 660001, Scotts Valley, CA 95067-0001

Borland may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1993 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. All other trademarks are the property of their respective owners.

Borland International, Inc.

1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001



RI 10 9 8 7 6 5 4 3 2 1

Contents

Introduction	1	Using SQL statements	26
How to use this book	1	Chapter 4	
Audience	1	SQL-enabled ObjectPAL	27
Where to find information	3	ObjectPAL SQL methods	27
Terms and conventions	3	Standard ObjectPAL methods that support SQL Link	29
Chapter 1		Standard ObjectPAL methods that do not support SQL	31
The SQL environment	5	Chapter 5	
Introduction	5	Using SQL Link: advanced topics	33
The database server	6	Table requirements for use of SQL Link capabilities	33
SQL	7	Unique row identification	33
SQL Link's role in the client/server environment	8	Enabling editing capabilities	33
Chapter 2		Blob access support	34
Your SQL-enabled desktop product	11	Defined row ordering	34
Preparing to use SQL Link	11	Locking and transaction support	34
Working in an SQL environment	12	Default transaction behavior	35
Desktop product window	12	Table-locking behavior	35
Using Table windows	13	Record-locking behavior	35
Using Form windows	13	Client-controlled transaction behavior	36
Using QBE	14	Transaction and locking errors for ObjectPAL programmers	37
Working with international language drivers	14	peDeadlock	37
The SQL Editor	15	peOptRecLockFail	37
SQL terminology	15	peOptRecLockRecDel	37
Chapter 3		Chapter 6	
Querying SQL data	17	Developing ObjectPAL applications	39
Introduction	17	Moving a local application to an SQL environment	39
Using QBE	18	Moving local tables	40
Typical QBE query operations	19	Re-binding forms	40
Retrieving all data	19	Changing table references	41
Retrieving specific columns	20	Changing index references	42
Retrieving specific rows	20	Using transactions in ObjectPAL applications	42
Specifying multiple search conditions	21	The SQL locking strategy	42
Specifying alternate search conditions	21	Applying transaction management principles	43
Specifying negative search conditions	22	"Snapshot" transactions	45
Eliminating duplicate entries	22	Generating the key at the end of the transaction	45
Pattern matching	23	Optimizing your SQL application	46
Specifying the order of rows	24	TCursors vs. queries in an application	46
QBE query processing	24	Lookup tables	47
Blocking QBE processing of queries	24	Stored procedures and triggers	47
QBE update queries	25		
Adding rows	25		
Changing values	25		
Removing rows	26		



Figures

Intro.1	How to use the Borland SQL Link documentation.	2
1.1	The network model.	6
1.2	The client/server model.	7
1.3	SQL Link's role in the client/server environment	9



Tables

1.1	Processing a sample transaction at the ATM.	8
1.2	Database Desktop functions with SQL Link	9
1.3	Paradox for Windows functions with SQL Link	10
2.1	Preparing to use SQL Link	11
2.2	Basic SQL terminology	15
3.1	SQL equivalents of typical desktop product queries	18
4.1	ObjectPAL SQL methods for the SQL object type	27
4.2	ObjectPAL methods for the SQL object type: syntax	28
4.3	SQL-enabled ObjectPAL methods for other object types.	28
4.4	SQL-enabled ObjectPAL methods for other object types: syntax	29
4.5	Standard ObjectPAL methods that support SQL Link	29
4.6	ObjectPAL methods that do not work with SQL Link	31



Introduction

Borland SQL Link 1.0 enables you to access data stored in an SQL database using the same tools with which you now access local data. This product supports Borland Paradox and Quattro Pro for Windows versions which are licensed for use with SQL Link.

Note Quattro Pro users access SQL data with SQL Link using the Database Desktop (DBD.EXE), not the Quattro Pro product itself (QPW.EXE). Database Desktop, which provides a look and feel compatible with Paradox for Windows, is included on the Quattro Pro installation disks.

SQL Link enables you to access SQL data in one of two ways:

- If you are a Paradox for Windows or a Quattro Pro for Windows user, you can access SQL data using tables and Query By Example (QBE). Paradox for Windows users can also use forms, reports, and SQL Link's SQL Editor window.
- If you are familiar with Paradox ObjectPAL, you can access SQL data by writing ObjectPAL applications and embedding SQL statements. This provides full access to all of the features and functions of database servers, including stored procedures, triggers, and data dictionaries.

How to use this book

This manual describes how to use SQL Link with your Borland desktop product. It describes how to use your Borland desktop product in its *SQL-enabled state*; that is, how to work with SQL data that resides on a server.

The *Borland SQL Link User's Guide* is meant to be used with the *Connecting to...* manual for your SQL database and your Borland desktop product user documentation.

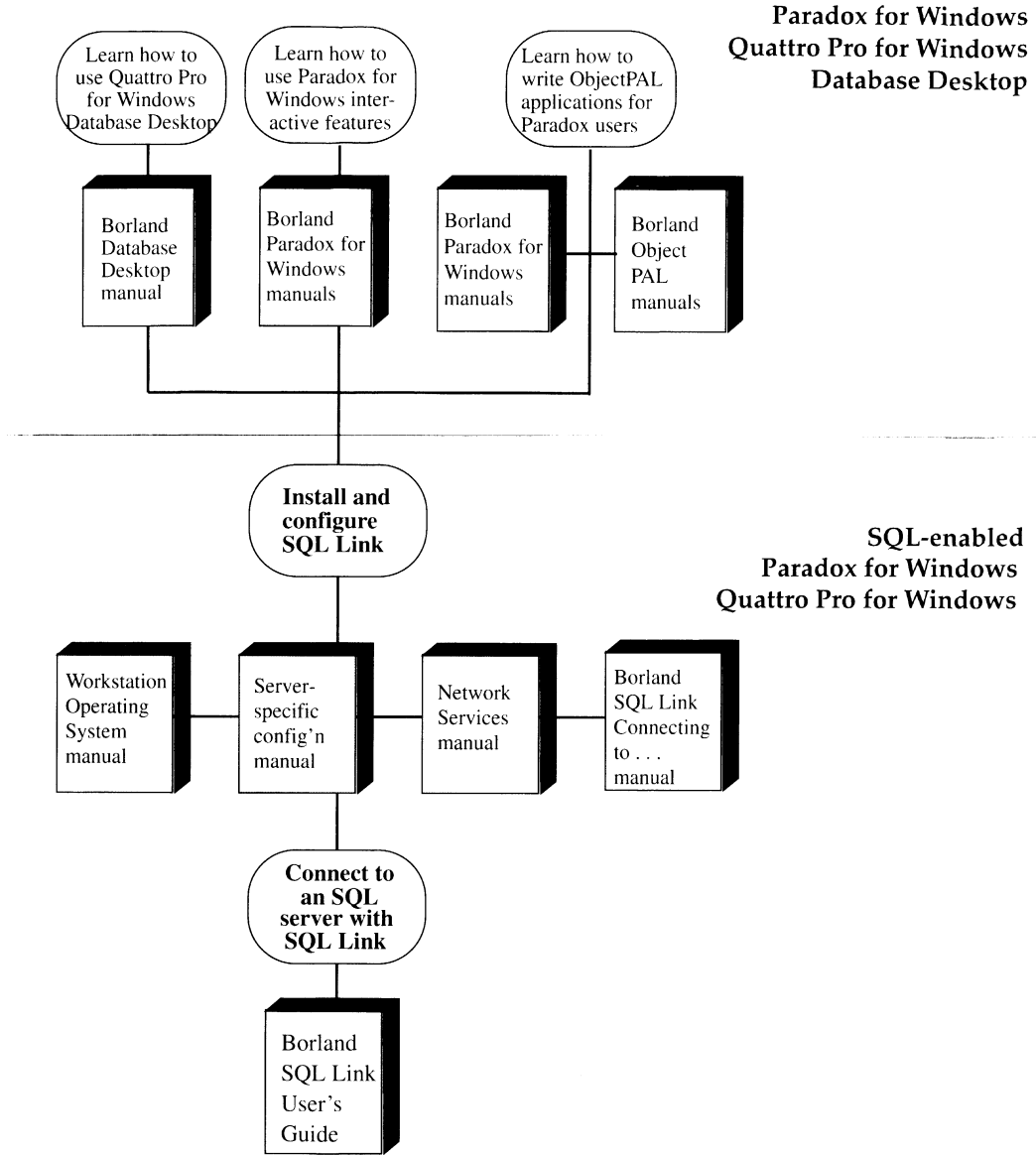
Audience

The *SQL Link User's Guide* is intended for use by three types of users:

- *Paradox and Quattro Pro users* who can use their desktop product with SQL Link to access information stored on SQL databases.
- *ObjectPAL programmers* who can use Paradox ObjectPAL language to:
 - Write scripts to access SQL databases.
 - Run SQL statements from within ObjectPAL applications.
 - Build SQL transaction processing techniques into their applications.
- *SQL users and programmers* who need to access SQL databases from within Paradox for Windows. (Quattro Pro for Windows has no SQL Editor or ObjectPAL capability.)

Figure 1 illustrates how each type of user can use their Borland documentation to work with SQL Link.

Figure Intro.1 How to use the Borland SQL Link documentation



Note This book does *not* attempt to teach you how to use your Borland desktop product, how to program in ObjectPAL, or how to construct SQL statements.

Where to find information

The following table shows where to find information in the *Borland SQL Link User's Guide*.

Topic	Where to find information
The client/server model	Chapter 1
Description of SQL	
Overview of using Borland desktop products with SQL Link	
Preparing to use SQL Link checklist	Chapter 2
Description of the SQL Editor	
Differences in SQL-enabled desktop product operation	
Querying data in SQL tables	Chapter 3
Using the SQL Editor window	
ObjectPAL methods available in SQL-enabled Paradox for Windows	Chapter 4
SQL Link advanced concepts	Chapter 5
ObjectPAL application development	Chapter 6

Terms and conventions

The Borland SQL Link manuals use special typefaces to help you distinguish between keys you press, names of objects, menu commands, and text you type. The following table lists these conventions.

Convention	Applies To	Examples
Bold type	Method names, desktop product status messages, and text that you type in	insertRecord Paradox displays the message Index error on key field . Type a:\install .
<i>Italic type</i>	Names of desktop product objects, glossary terms, variables, emphasized words Occasionally used for emphasis at the beginnings of bulleted paragraphs, to tie in with the preceding paragraph	<i>Answer table</i> , <i>searchButton</i> , <i>searchVal</i> <i>The drive (or directory) from which you are installing SQL Link</i> . Usually, this will be either drive A or drive B.
ALL CAPS	DOS files and directories, reserved words, operators, types of queries	PARADOX.EXE, C:\WINDOWS, CREATE
Initial Caps	Applications, fields, menu commands	Sample application, Price field, Form View Data command
<i>Keycap Font</i>	Keys on your computer keyboard	F1 , Enter
Monospaced font	Code examples, ObjectPAL code	<code>myTable.open("sites.db")</code>

The following table lists conventions used for ObjectPAL syntax.

ObjectPAL Convention	Element	Examples	Meaning
Normal font	Keyword	setPosition	Type exactly as shown.
<i>Italic</i>	Fill-in	<i>tablevar</i>	Replace with an expression.
{ } (braces and bar)	Choice	{ Yes No }	You <i>must</i> choose one of the elements separated by the vertical bar.
[] (brackets)	Optional	[, <i>tableVar2</i> ELSE]	You <i>can</i> choose whether or not to include this.
* (asterisk)	Repeat	[, <i>tableVar2</i>] *	You can repeat this argument.

The SQL environment

This chapter provides background information on concepts that are unique to SQL, and describes the role of SQL Link in a client/server environment. By understanding these basic terms and concepts, you will be able to use your Borland desktop product more effectively to work with data on your SQL server.

If you are already familiar with SQL and SQL servers, you may only need to browse through this chapter.

Introduction

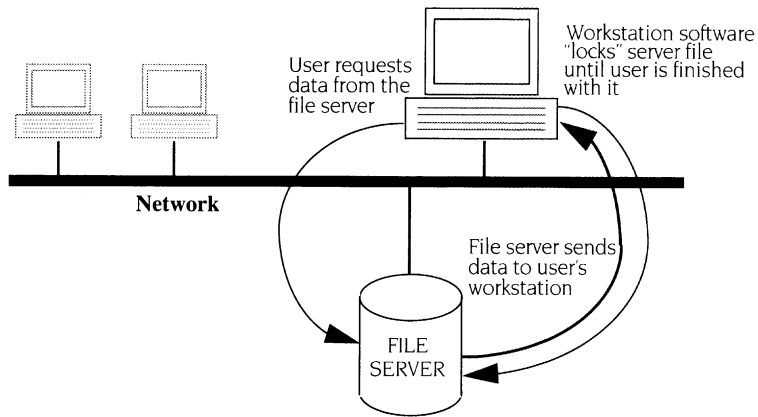
If you have ever worked in a setting where you had to share access to files or printers, you are probably familiar with the idea of a *local area network*, or LAN. A LAN enables workstation users to share files, software, and printer resources stored on dedicated machines called *servers*. Workstations connect to network servers through a system of cabling, communications hardware, and software. In large user populations, two or more LANs can connect through *gateways* to form *wide area networks*, or WANs.

In a network environment, your workstation uses the network server in much the same way as it uses its own hard disk. If your workstation needs access to data stored on the server's hard disk it requests that data from the server. The server sends the requested data over the network and back to your workstation where it is processed locally.

The network server differs from the workstation in that server data can be accessed by more than one user at the same time. For example, you may want to read a file at the same time as another user is trying to edit it. To guard against users making conflicting changes, the application on the first workstation must lock the data in use so that no one else can modify it.

Figure 1.1 shows how this works.

Figure 1.1 The network model.



Since database servers use similar types of networking hardware and software, it is logical to assume that database servers operate just like other types of network servers. However, as discussed in the following section, database servers operate differently from other types of network servers.

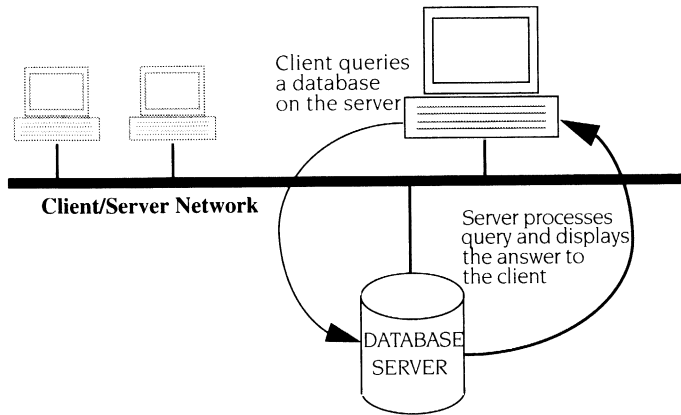
The database server

A *database server* is a computer that processes high-level database requests. Database servers operate differently from other types of network servers. Other types of network servers act mainly as passive data "libraries," with all of the processing being done on the user's workstation. Database servers are active with most processing being done on the database server itself. If your workstation needs access to data stored in a database server, you *query* the server directly. The database server processes the query itself and sends only the answer over the network and back to your workstation.

Since the processing is performed at the server and not at the workstation that originated the request, the workstation becomes a *client* of the database server. In a *client/server system*, multiple clients (users) request the services of the *database server* through the use of *client applications* such as Paradox for Windows or the Quattro Pro Database Desktop.

Figure 1.2 shows how this works.

Figure 1.2 The client/server model.



In a client/server environment, the type of database you generally find on a server is called a *relational database*. The types of operations you can perform with a relational database server are governed by the rules for *relational database management systems* (RDBMS).

For further information about RDBMS, many outside sources are available. For example, C.J. Date's *An Introduction to Database Systems* (Addison-Wesley, Reading, Massachusetts, 1983).

SQL

What we now call SQL is actually a descendant of SEQUEL (or Structured English QUery Language), which was designed at IBM over twenty years ago. The first commercial RDBMS using SQL appeared in 1981, and SQL is now the standard for network queries across different hardware and software platforms.

Note Many people still pronounce SQL as "sequel," but in this book--to avoid confusion with SEQUEL and other copyrighted IBM product names--we use the term as if it is pronounced "ess-q-ell."

SQL is actually a *sublanguage* designed to be embedded in an application programming language. It is so flexible that it can be used for data manipulation (retrieval and modification), data definition (creating the tables that describe the nature of each piece of data in a row or column), and data administration.

SQL has many advantages as a tool for managing relational databases. The two most important are:

- It is supported by many vendors. SQL programs are available for almost any server machine in your network. The only requirement is that the communications equipment on either end uses the same SQL dialect.
- It is easy to learn. Although SQL is a sophisticated application development language, it uses simple English verbs (such as SELECT, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK) to describe a desired operation.

SQL database servers handle requests in logical units of work called *transactions*. A transaction is a group of related operations that must all be performed successfully before the RDBMS will finalize any changes to the database. Transaction processing on database servers ensures that your processing requests are appropriate to the current state of your data.

In SQL, all transactions can be explicitly ended with a command to either accept or discard the changes. Once you are satisfied that no errors occurred during the transaction, you can end that transaction with a COMMIT command. The database then changes to reflect the operations you have just performed. If an error occurs, you can abandon the changes with the ROLLBACK command.

For example, suppose you want to withdraw money from your checking account using an automated teller machine (ATM). Table 1.1 shows the actions you must take and the operations the ATM must complete before you can receive your money.

Table 1.1 Processing a sample transaction at the ATM

Action you perform	Operation the ATM must complete
1. Insert ATM card.	1. Determine the number of the target bank account.
2. Enter personal identification code.	2. Verify user's permission to access target account.
3. Identify desired transaction and enter amount of money to withdraw.	3. Check balance of target account to make sure desired transaction will not bring the account balance below \$0.00. If sufficient funds exist, the transaction is committed, the withdrawal amount subtracted from the balance, the new balance posted, and the money dispensed. If sufficient funds do not exist, the transaction is rolled back.
4. If the transaction succeeds, remove money from ATM window. If the transaction fails, remove card.	

To receive your money, the ATM must complete each operation associated with the transaction. These operations constitute a single transaction, because they cannot be logically separated. If each operation succeeds, then the entire transaction succeeds. If any of the operations fail, then the entire transaction fails.

SQL Link's role in the client/server environment

One important aspect of communications with a database server is that all queries must be framed in appropriate SQL dialect to be understood. However, Borland SQL Link enables you to access data stored on SQL database servers without having to learn SQL.

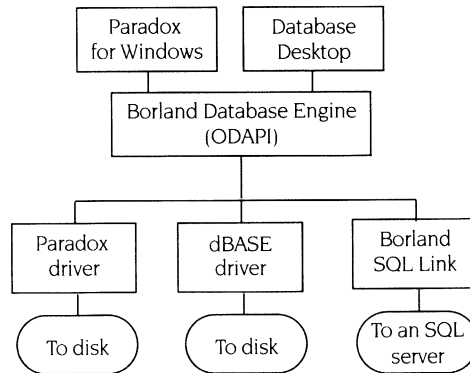
SQL Link is designed for Paradox for Windows and Quattro Pro for Windows users who are unfamiliar with SQL, but need to access SQL data. SQL Link enables you to use your Borland desktop product to access an SQL server the same way you use it to access a local Paradox or dBASE database. If you are familiar with Paradox for Windows or Quattro Pro for Windows, you can work with SQL data using Table windows or the QBE environment. If you have Paradox for Windows, you can also access an SQL server using forms, reports, or the SQL Link SQL Editor. Paradox ObjectPAL programmers can access an SQL server by writing ObjectPAL applications.

Note Quattro Pro users use SQL Link with the Quattro Pro Database Desktop (DBD.EXE), and not the Quattro Pro product itself (QPW.EXE). The Database Desktop provides a look and feel compatible with Paradox for Windows.

Regardless of which method you choose, SQL Link enables your connection to the database server, translates your query into the appropriate SQL dialect, and passes it to the SQL database. When processing is complete, the SQL database returns the answer to the client in a format that your Borland desktop product can display.

Figure 1.3 shows how this works.

Figure 1.3 SQL Link's role in the client/server environment



Note In some cases, queries to SQL databases will be processed under QBE rules. For further information, see Chapter 3.

The following tables describe which Borland desktop product operations can be performed with SQL Link and notes where they might be different in a client/server environment.

Table 1.2 Database Desktop functions with SQL Link

Paradox function	Notes
Use tables to open and view an SQL table.	Tables on SQL data are read-only; pressing F9 to edit the table results in a message saying the table is read-only.
Use QBE to query and update SQL table data.	
Use the Alias Manager to manage multiple connections to SQL databases.	

Table 1.3 Paradox for Windows functions with SQL Link

Paradox function	Notes
Use tables to open and view an SQL table.	Tables on SQL data are read-only; pressing F9 to edit the table results in a message saying the table is read-only. To edit SQL table data you may use a form, QBE, or the SQL Editor. Commands normally available from File Utilities are not enabled for SQL tables.
Use forms to open and view an SQL table, or create forms to edit SQL table data.	
Create cross tabs in forms to analyze relationships in SQL table data.	
Create reports to view and print SQL table data.	
Use QBE to query and update SQL table data.	
Design ObjectPAL applications that access SQL table data, including some that support transaction processing.	
Use pass-through SQL to send SQL statements directly to your database server to be executed at the database server. The data returned from the server is placed in a Paradox or dBASE <i>Answer</i> table.	
Use the SQL Editor to view, modify, and execute an SQL statement that is equivalent to a query you created with QBE.	At any time (whether or not you are in the Query window) you can also use the SQL Editor to create, load, modify, or execute an SQL statement. You can also save the SQL statement to a disk file.
Use the Alias Manager to manage multiple connections to SQL databases.	

The following chapter provides a checklist to help you prepare for using SQL Link and describes new features for your SQL-enabled Borland desktop product.

For detailed information on how to query an SQL database with SQL Link, see Chapter 3.

For information about ObjectPAL methods that work in an SQL environment, see Chapter 4.

Your SQL-enabled desktop product

This chapter describes what to expect when your product becomes SQL-enabled. It includes a checklist that summarizes the SQL Link installation and configuration process and a brief description of a new feature added with SQL Link: the SQL Editor.

Note The SQL Editor is not available in SQL-enabled Database Desktop 1.1.

For examples of how to query and update SQL data with SQL Link, see Chapter 3.

For information about ObjectPAL methods that work in an SQL environment, see Chapter 4.

Note Be sure you have already installed your Borland desktop product, as described in your desktop product documentation.

Preparing to use SQL Link

Table 2.1 summarizes the steps you must complete before you can begin to use SQL Link with your Borland desktop product.

Table 2.1 Preparing to use SQL Link

Action to complete	See . . .
Before SQL Link installation	
Ensure that you have met your SQL server software prerequisites.	Server-specific <i>Connecting to...</i> book, Table 1.1
Ensure that you have met your client workstation requirements (includes hardware and software requirements for your Borland desktop product).	Your Borland desktop product installation documentation Server-specific <i>Connecting to...</i> book, Table 1.2
Ensure that you can access the SQL database. You need a valid user identification and password on the SQL server, and at least Read access privileges for the SQL database.	Your Database Administrator

Action to complete

Determine where you want to install the SQL Link files. (This is usually the same directory as the desktop product ODAPI [database engine] files. The default directory is C:\ODAPI).

InterBase users only:

Determine the directory where you want to store the InterBase message and log files (Default: C:\INTERBAS).

Identify what kind of network software you use to access the database server (LAN Workplace, Network File System, or File Transfer Protocol).

SQL Link Installation

See . . .

Your Borland desktop product installation documentation, or search for the location of ODAPI.* on the workstation hard disk

Your Database Administrator

Server-specific *Connecting to...* book, Chapter 1

After SQL Link Installation

Specify default driver settings in ODAPI.CFG file

Server-specific *Connecting to...* book, Chapter 2

Create at least one Alias to the SQL server

Server-specific *Connecting to...* book, Chapter 2

Working in an SQL environment

As noted in Chapter 1 (see Tables 1.1 and 1.2), adding SQL Link to an existing Borland desktop product gives you some new capabilities.

This section discusses how your desktop product operation changes when you access an SQL server and briefly describes the SQL Editor.

For information about new SQL methods for ObjectPAL programmers, see Chapter 4.

Desktop product window

When the desktop product is SQL-enabled, you experience the following general changes in the desktop product window:

- *SQL hourglass.* Whenever you access the SQL server you see the SQL hourglass.
- *Working directories.* Borland desktop products let you set up a working directory to store all the tools you use whenever you work with local or shared data. However, since you cannot store objects such as documents, queries, reports, and forms on SQL servers, you cannot set an SQL server as the location for your working directory.

TIP You might want to set up a local folder to hold all the forms, reports, .TVS files, and queries you use when you work with a particular SQL database. Once you connect to that database you can then open the local folder and put your working tools at your fingertips. You can also easily apply the tools across other SQL databases.

- *Refreshing data displays.* In a non-SQL environment, as soon as one user makes a change to a shared database your Borland desktop product automatically updates all users' data. However, when operating in SQL-enabled mode this does not occur. To update the active window on your screen when working with SQL data, press **Ctrl+F3** periodically. **Ctrl+F3** shows any updates made to a table while you are viewing it.

Note: **Ctrl+F3** does not update views of SQL tables without a defined row ordering. For further information, see Chapter 5.

Using Table windows

When the desktop product is SQL-enabled, you experience the following general changes in how Table windows work:

- Table windows are read-only for SQL data. To change SQL table data from your Borland desktop product, use a form or QBE; if you use Paradox for Windows you can also use update queries through the SQL Editor window.
- Table windows of SQL data do not display record numbers; the thumb is always in the center of the vertical scroll bar.
- Since (unlike Paradox and dBASE) SQL servers support Date/Time fields, there is a new dialog box for Timestamp.
- Because of differences in how indexing functions in an SQL environment, there are minor differences in the Order/Range dialog box.
- Table properties for an SQL table are stored in a file with the extension *.TVS*. This helps distinguish them from Table windows property files for Paradox tables (which are stored in files that end in *.TV*) and dBASE tables (which are stored in files that end in *.TVF*). Although *.TV* and *.TVF* files are automatically stored as *tablename.ext*, *.TVS* files need to be explicitly named by the user. The functionality of *.TVS* files is otherwise the same.
- If you try to view an SQL table when someone else is editing data, you may have to wait until the other user is finished editing. This is because your SQL server may wait for the other user's edit to complete. (For further information about record locking in SQL, see Chapter 5.)

Using Form windows

When Paradox for Windows is SQL-enabled, you experience the following general changes in how forms work:

- SQL tables in forms are read-only by default. If you want to use a form to modify SQL data, you need to ensure that you can write changes to the table. To do this:
 - 1 Select Form | Data Model or click the Data Model button on your desktop product SpeedBar.
 - 2 Inspect each table in the model.
 - 3 Uncheck read-only for each table item you want to be able to modify.Your changes take effect immediately.
- Because of differences in how indexing functions in an SQL environment, there are minor differences in the Order/Range dialog box.
- Since record locking in SQL Link follows different rules than in Paradox or dBASE, editing, and posting of changes to SQL data is different. (For further information about record locking in SQL, see Chapter 5.)

Using QBE

The characteristic behavior of SQL update queries dictates that updates are performed either completely or not at all. When you use QBE to perform updates on SQL data your Borland desktop product does not generate any of the following auxiliary tables:

CHANGED.DB
INSERTED.DB
DELETED.DB
ERRCHG.DB
ERRINS.DB
ERRDEL.DB

For detailed information on how to use QBE to query and update SQL data, see Chapter 3.

Working with international language drivers

Servers running on different systems use conventions called *character sets* (or *code pages*) to determine how to encode alphabetic data. If you operate in a non-English language environment, your Borland desktop product may use a different character set than the one used by the SQL server. When the character set at your desktop does not match the character set at the SQL server, passing alphabetic data between the two may cause either (or both) of the following results:

- Data displays incorrectly on your desktop
- The wrong characters are recorded in the SQL database

To prevent this from happening, SQL Link provides language drivers to convert character data between your desktop product character set and the SQL server character set. This ensures that characters from your server display correctly on your desktop, and that data you enter in the SQL server database is transmitted reliably.

Language drivers also contain information on *sort order* and *uppercasing* conventions used by your SQL server. Whenever a query to an SQL database is processed under QBE rules, the desktop product language driver is used in evaluating character ranges for sorting. If the sort order and uppercasing conventions at your desktop do not match the conventions used at the SQL server, your desktop product may display inconsistent results.

If your SQL database uses extended character sets, make sure the Alias you use to access the SQL server specifies the correct SQL Link language driver. Choose a language driver that uses the same sort order and uppercasing conventions used by the SQL server.

Note A different sort order can cause the selection of a different set of records. If you cannot find an SQL Link language driver that uses the same sort order as your SQL server, you may want to modify your Alias SQLQRYMODE entry to prevent processing of queries under QBE rules. For further information, see Chapter 3.

The SQL Editor



Once you configure SQL Link for use with Paradox for Windows, the Desktop window SpeedBar includes a button labeled "SQL." This button provides the SQL Editor function.

When you use QBE to query a table in an SQL database, SQL Link attempts to translate your query to an equivalent SQL statement and pass it to the SQL server. If successful, the server processes your query, then passes the answer set back to you through SQL Link. The SQL Editor enables you to view the equivalent SQL statement for the query at any time during query construction, or after it is processed.

It is possible to express queries using QBE that do not have a direct SQL equivalent. In such cases, the SQL Editor display is blank. (See "Query processing," in Chapter 3.)

For examples of how to use the SQL Editor with QBE to build queries in pass-through SQL, see Chapter 3.

SQL terminology

Since this book is intended for use solely in an SQL environment, it uses traditional SQL terminology to describe the structure of that environment. Table 2.2 defines the terms used to describe SQL data and shows how those terms are used in other Borland desktop product documentation.

Table 2.2 Basic SQL terminology

SQL	Paradox for Windows	Database Desktop	Definition
Table	Table	Table	A structure of rows (records) and columns (fields) that contains information.
Row	Record	Record	A group of columns (fields) in a table that contain related information about a single record.
Column	Field	Field	A category of information (column) in a table that cuts across all rows in the table.

Querying SQL data

Query By Example (QBE) provides you with a graphical format that helps you show what kind of information you want in your *Answer* table. When you use QBE to query a table in an SQL database, SQL Link attempts to translate your query to an equivalent SQL statement and pass it to the SQL server. If successful, the server processes your query, then passes the answer set back to you through SQL Link. The SQL Editor enables you to view the equivalent SQL statement for the query at any time during query construction, or after it is processed.

This chapter describes how to use QBE and the SQL Editor to view, access, retrieve, and change SQL data. It is intended for use by all users of SQL-enabled Borland desktop products, even if your product does not supply the SQL Editor function.

Be sure you have already created an Alias for the SQL server you want to access, as described in Chapter 2 of your server-specific *Connecting to...* manual.

For general information on SQL-enabled ObjectPAL methods, see Chapter 4.

Introduction

As noted in Chapter 1 (see Tables 1.1 and 1.2), adding SQL Link to an existing Borland desktop product gives you some new capabilities. Many of the differences in your SQL-enabled desktop product are described in Chapter 2.



Once you configure SQL Link for use with Paradox for Windows, the Desktop window SpeedBar includes a button labeled "SQL." The SQL button activates the SQL Editor function, which enables you to view the equivalent SQL statement for queries you construct in the QBE environment.

Note If the SQL database does not support an equivalent SQL statement for a QBE query, the SQL Editor display is blank and the query is processed in the QBE environment. For further information, see "Query processing," later in this chapter.

To view the SQL “translation” for a query you constructed using QBE:

- 1 Connect to the SQL database.
- 2 Use QBE to construct a query to the SQL database.
- 3 Open the SQL Editor window in either of the following ways:
 - Choose Query | Show SQL from your Borland Desktop application menu.
 - Click the SQL SpeedBar button.

Depending on the type of query you just created, the SQL translation will be one of the following types of statements:

Table 3.1 SQL equivalents of typical desktop product queries

Desired result of query	Equivalent SQL statement
Display specific data	SELECT
Add new data	INSERT
Change existing data	UPDATE
Remove existing data	DELETE

Programmers familiar with SQL can use the SQL Editor window to directly enter, execute, or save an SQL statement. The SQL server executes such statements without any involvement by QBE. If you press the SQL button when there is no active query on the Desktop, the SQL Editor assumes you want to edit an existing SQL file on your workstation or write a new SQL query.

The following section describes using QBE to frame a query to an SQL server, and viewing that query through the SQL Editor window.

For further information on using pass-through SQL, see “Using SQL statements,” later in this chapter.

Using QBE

Querying an SQL table works exactly the same way as querying a local table in your Borland desktop product:

- 1 Choose File | New | Query from your desktop product’s Main menu.
- 2 Select the Alias to the SQL server you want to query.
- 3 Select the desired SQL table.
- 4 Fill out the Query window, specifying criteria to select the data you want to retrieve.
- 5 Press *F8* (Run) or click the Run Query SpeedBar button to process your query.

SQL Link also supports the use of queries that join SQL tables with local tables, or SQL tables from different SQL databases (heterogeneous queries). Heterogeneous queries are always processed according to QBE rules.

Typical QBE query operations

This subsection describes how to perform typical QBE query operations on SQL data. For each type of query, you see the filled-out Query window and the equivalent SQL statement.

Note You cannot interrupt a query while it is processing as long as the SQL hourglass is visible. The size of the SQL table determines query retrieval time.

Once you become familiar with the syntax of SQL queries, you may prefer to use the SQL Editor to write SQL statements and send them directly to your server. This type of query is always processed by the rules of your SQL server. For more information, see “Using pass-through SQL,” later in this chapter.

Note QBE queries are automatically under transaction control. However, if you run the SQL equivalent of a QBE query, those SQL statements are not under automatic transaction control. Transactions must be explicitly begun and either committed or rolled back.

Retrieving all data

To retrieve all rows and columns from a particular table, check all the columns in the Query window:

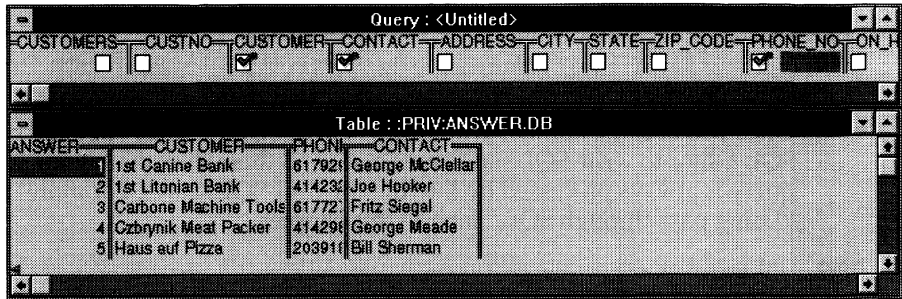
ANSWER	CUSTNO	CUSTOMER	CONTACT	ADDRESS	CITY	STATE	ZIP_CODE	PHONE_NO	ON_HOLD
1	100.00	Thomas Nove	George Thor	Tremont St	Boston	MA	02108	6179291181	
2	101.00	Scott Bros Au	Winfield Sco	sweet Lane	Randolph	MA	02368	6179632219	*
3	102.00	J.L. Music Corp	Ambrose Bur	raba Ave S	Chicago	IL	60649	3123429882	
4	103.00	The Groton C	Phil Sheridar	Great Road	Groton	MA	01450	5086436627	
5	104.00	Pope & Saun	John Pope	inole Drive	alachiola	FL	32320	3052983744	*

SQL Link translates the above query into:

```
SELECT DISTINCT CUSINO, CUSTOMER, CONTACT, ADDRESS, CITY, STATE, ZIP_CODE,  
PHONE_NO, ON_HOLD  
FROM CUSTOMERS  
ORDER BY CUSTNO, CUSTOMER, CONTACT, ADDRESS, CITY, STATE, ZIP_CODE,  
PHONE_NO, ON_HOLD
```

Retrieving specific columns

To retrieve only specific columns from a particular table, check only the columns of interest in the Query window:



The screenshot shows a query window titled "Query : <Untitled>". The columns selected in the query are CUSTOMER, CONTACT, and PHONE_NO. Below the query window, a table titled "Table : :PRIV:ANSWER.DB" displays the results of the query. The table has five rows of data.

ANSWER	CUSTOMER	PHONE	CONTACT
1	1st Canine Bank	617921	George McClellan
2	1st Litonian Bank	414230	Joe Hooker
3	Carbone Machine Tools	61772	Fritz Siegal
4	Czbrnyk Meat Packer	414290	George Meade
5	Haus auf Pizza	203911	Bill Sherman

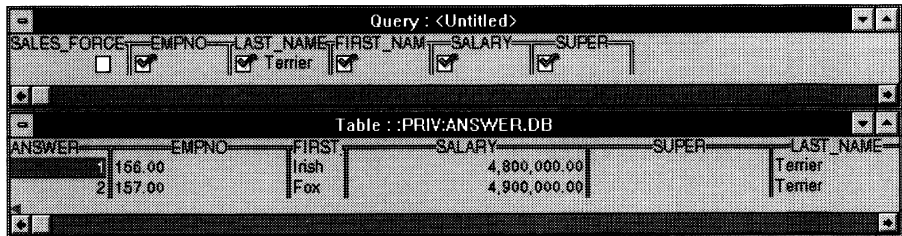
The SQL equivalent of the above query is:

```
SELECT DISTINCT CUSTOMER, CONTACT, PHONE_NO
FROM CUSTOMERS
ORDER BY CUSTOMER, CONTACT, PHONE_NO
```

Retrieving specific rows

To retrieve only specific rows from a particular table, check all the columns in the Query window but give specific search criteria in at least one of the columns. This retrieves all the rows that match your search criteria as complete records, all columns included.

To retrieve only specific rows and specific columns, check only the columns of interest *and* give specific search criteria in at least one of the checked columns. This retrieves all the rows that match your search criteria and displays only the columns of interest.



The screenshot shows a query window titled "Query : <Untitled>". The columns selected in the query are EMPNO, LAST_NAME, FIRST_NAME, SALARY, and SUPER. A search criteria "Terrier" is entered in the LAST_NAME field. Below the query window, a table titled "Table : :PRIV:ANSWER.DB" displays the results of the query. The table has two rows of data.

ANSWER	EMPNO	FIRST	SALARY	SUPER	LAST_NAME
1	160.00	Insh	4,800,000.00		Terrier
2	157.00	Fox	4,900,000.00		Terrier

The SQL equivalent of the above query is:

```
SELECT DISTINCT EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
FROM SALES_FORCE
WHERE (LAST_NAME = 'Terrier')
ORDER BY EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
```

Specifying multiple search conditions

To specify multiple search criteria, combine the techniques for retrieving rows and columns. The SQL server retrieves only those rows that match *all* of the search criteria.

The screenshot shows a query window titled "Query : <Untitled>". The query is defined on the "SALES_FORCE" table with the following columns: EMPNO, LAST_NAME, FIRST_NAME, SALARY, and SUPER. The search criteria are: LAST_NAME = 'Terrier' and FIRST_NAME = 'Fox'. The results are displayed in a table titled "Table : :PRIV:ANSWER.DDB".

ANSWER	EMPNO	FIRST	SALARY	SUPER	LAST_NAME
1	157.00	Fox	4,900,000.00		Terrier

The SQL equivalent of the above query is:

```
SELECT DISTINCT EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
FROM SALES_FORCE
WHERE (LAST_NAME = 'Terrier')
AND (FIRST_NAME = 'Fox')
ORDER BY EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
```

Specifying alternate search conditions

Another way to specify alternate search criteria is to combine criteria on separate rows of the query. This type of search retrieves those rows that match either (or any) of the search criteria.

The screenshot shows a query window titled "Query : <Untitled>". The query is defined on the "SALES_FORCE" table with the following columns: EMPNO, LAST_NAME, FIRST_NAME, SALARY, and SUPER. The search criteria are: LAST_NAME = 'Terrier' and SALARY = 48000. The results are displayed in a table titled "Table : :PRIV:ANSWER.DDB".

ANSWER	EMPNO	FIRST	SALARY	SUPER	LAST_NAME
1	142.00	Alex	4,900,000.00	122.00	Springer
2	156.00	Irish	4,900,000.00		Terrier
3	157.00	Fox	4,900,000.00		Terrier

The SQL equivalent of the above query is:

```
SELECT DISTINCT EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
FROM SALES_FORCE
WHERE (LAST_NAME = 'Terrier')
OR (SALARY = 48000)
ORDER BY EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
```

Specifying negative search conditions

To exclude certain rows from the *Answer* table, use the QBE NOT operator. The NOT operator acts like the SQL inequality operator <> or the SQL NOT operator. It returns only those rows that do *not* meet the search criteria.

Query : <Untitled>						
SALES_FORCE	EMPNO	LAST_NAME	FIRST_NAME	SALARY	SUPER	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> Terrier	<input checked="" type="checkbox"/> NOT Irish	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Table : :PRIV:ANSWER.DB						
ANSWER	EMPNO	FIRST	SALARY	SUPER	LAST_NAME	
	157.00	Fox	4,900,000.00		Terrier	

The SQL equivalent of the above query is:

```
SELECT DISTINCT EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
FROM SALES_FORCE
WHERE (LAST_NAME = 'Terrier')
AND (FIRST_NAME <> 'Irish')
ORDER BY EMPNO, LAST_NAME, FIRST_NAME, SALARY, SUPER
```

Eliminating duplicate entries

Occasionally a database may include duplicate entries. When you narrow your search to only specific criteria, the *Answer* table may include all rows that match those criteria, even if they contain some duplicate entries.

To exclude duplicate entries from the *Answer* table, use the QBE Check (F6) operator. The Check operator acts like the SQL DISTINCT keyword. When you use Check, the result contains fewer rows and no duplicates.

Using Check may also improve database performance.

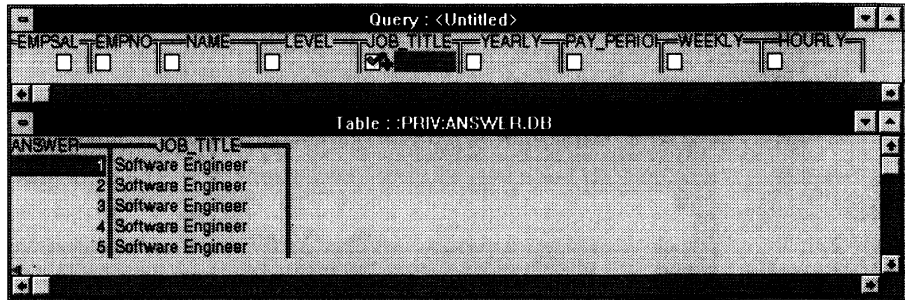
Query : <Untitled>								
EMPMSAL	EMPNO	NAME	LEVEL	JOB_TITLE	YEARLY	PAY_PERIOD	WEEKLY	HOURLY
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Table : :PRIV:ANSWER.DB	
ANSWER	JOB_TITLE
1	Administrative Clerk
2	Director
3	Educator
4	Manager
5	President
6	Sales Support
7	Salesman
8	Senior Software Engine
9	Software Engineer

The SQL equivalent of the above query is:

```
SELECT DISTINCT JOB_TITLE
FROM EMPMSAL
ORDER BY JOB_TITLE
```

If you want to return all rows that match the search conditions, including duplicates, use Check+ (Shift+F6 until you see the Check+ sign) instead of Check (F6).

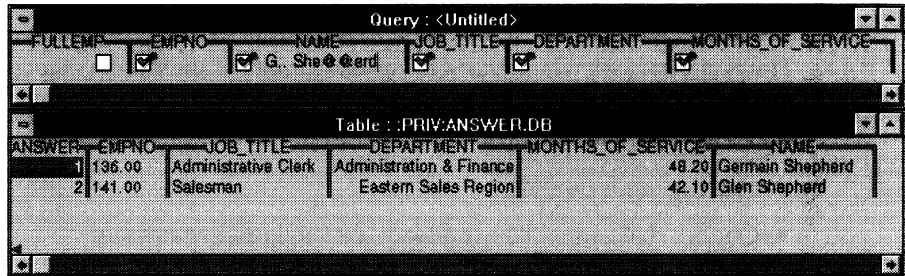


The equivalent SQL statement for this query does not include the DISTINCT keyword:

```
SELECT JOB_TITLE
FROM EMP
```

Pattern matching

QBE uses a case-insensitive scheme for searching through patterns in data, and some SQL servers only support case-sensitive pattern matching. If your SQL server supports QBE-style pattern matching, a query with pattern-matching is performed at the server. QBE wild cards work just like SQL wild cards; substitute an @ operator for any single character or a .. operator for any string of zero or more characters.

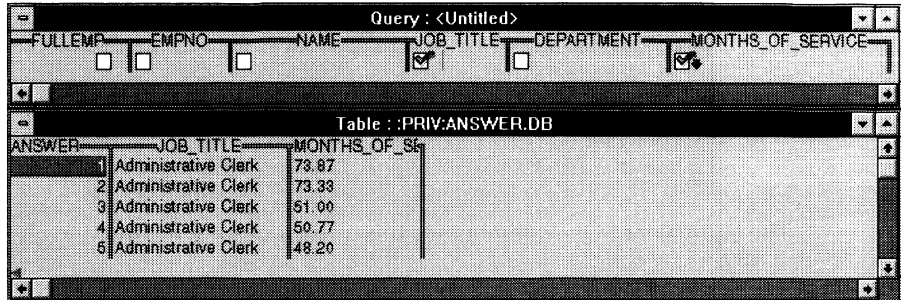


The equivalent SQL statement translates the QBE @ operator into the SQL _ (underscore) character and the QBE .. operator into the SQL % character.

```
SELECT DISTINCT EMPNO, NAME, JOB_TITLE, DEPARTMENT, MONTHS_OF_SERVICE FROM
EMP
WHERE (NAME LIKE '%She_erd')
ORDER BY EMPNO, NAME, JOB_TITLE, DEPARTMENT, MONTHS_OF_SERVICE
```

Specifying the order of rows

When you use QBE's Check function to specify columns you want in an *Answer* table, your desktop product automatically sorts the rows to be retrieved in ascending order. If you want to sort in descending order, mark the appropriate column with a Check Descending.



SQL Link translates the query using the keywords ORDER BY and DESC (descending order), as needed.

```
SELECT DISTINCT JOB_TITLE, MONTHS_OF_SERVICE
FROM FULLEMP
ORDER BY JOB_TITLE, MONTHS_OF_SERVICE DESC
```

QBE query processing

In most circumstances, queries to SQL databases are processed on the SQL server. However, in cases where the SQL server cannot (or should not) process a query, the desktop product software processes the query locally.

Types of SQL database queries that get processed locally include the following:

- Heterogenous queries
- Queries that cannot be expressed as a single SQL statement
- Queries that the SQL server does not support

While a query is processing, SQL Link always displays the SQL hourglass. If the SQL server is not going to process a query it downloads the necessary data to the desktop product for local processing and displays the message "Query is processing locally."

Blocking QBE processing of queries

In some cases, the result of a query executed completely on an SQL server will differ from that of a query executed locally. For example, QBE does a case-sensitive search on character fields. If the server does not support case-sensitive searches on a character field, then selecting on criteria such as ">A" will produce a different answer set depending on where the query is processed.

If you want to make sure that all queries originating from QBE are processed according to the rules of the SQL server, you can configure the SQL Link driver to block QBE

processing of queries. For example, you may want to force remote processing if you are using a non-English language driver that does not properly mimic the SQL server's conventions for sorting and uppercasing.

To do so, use the ODAPI Configuration Utility to modify the SQL database Alias, setting SQLQRYMODE to SERVER. (For a full description of how to modify an existing Alias, see Chapter 2 of your server-specific *Connecting to...* manual.)

The new SQLQRYMODE value takes effect the next time you start your Borland desktop product.

QBE update queries

QBE's Query window also functions as an updating device, enabling you to add, delete, and change SQL records. SQL Link translates these "update queries" into SQL INSERT, DELETE, and UPDATE statements.

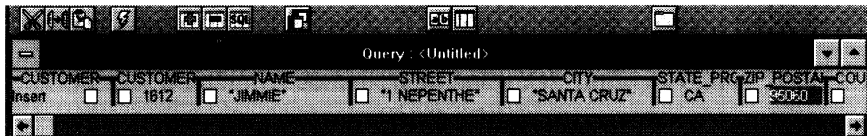
Updating an SQL table works exactly the same way as updating a local table in your Borland desktop product, except that no auxiliary tables are produced.

This section describes how to use QBE to update an SQL database. For each type of query, you see the filled-out Query window and the equivalent SQL statement.

Note You cannot interrupt a query while it is processing as long as the SQL hourglass is visible. The size of the SQL table determines query retrieval time.

Adding rows

To add a new row of data to an SQL table, use QBE's Insert function. Tab through the columns in the Query window, entering the desired information. When you are finished typing new information, run the query.



SQL Link translates the above operation into the SQL statement:

```
INSERT INTO CUSTOMER (CUSTOMER_NO, NAME, STREET, CITY, STATE_PROV, ZIP_POSTAL_CODE)
VALUES (1612, 'JIMMIE', '1 NEPENTHE', 'SANTA CRUZ', 'CA', 95060)
```

Changing values

To change the values of one or more items in an SQL table, use QBE's CHANGETO function. Enter selection criteria that identify the rows where you want the change to occur, then define the desired change. Be sure the CHANGETO operator is on the same line in the query window as the selection conditions. (Do not check any of the fields on this line of the Query window.)

When you are finished typing CHANGETO information, run the query.



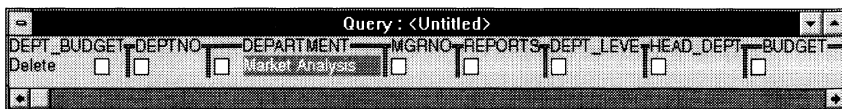
SQL Link translates the above operation into the SQL statement:

```
UPDATE EMP_SAL
SET YEARLY = (YEARLY + 750)
WHERE (JOB_TITLE = 'Support Engineer')
```

Removing rows

To remove one or more rows from an SQL table, use QBE's DELETE function. Enter any selection condition to select the rows to be deleted. (If you do not enter any selection conditions, all rows are deleted from the table.)

When you are finished typing selection conditions, run the query.



SQL Link translates the above operation into the SQL statement:

```
DELETE FROM DEPT_BUDGET
WHERE (DEPARTMENT = 'Market Analysis')
```

Using SQL statements

The SQL Editor window also enables you to pass SQL statements directly to the SQL server. You can save the SQL statement to a disk file (the SQL Editor automatically saves the file with the extension *.SQL*), and then later load, modify, or execute it.

Caution When you execute an SQL statement, Paradox sends it directly to your server for processing. Any error or syntax checking is performed by the SQL server.

Depending on what you want to do, you can open the SQL Editor window in a number of ways:

- To open (and edit or execute) an existing *.SQL* file, choose File | Open | SQL File, click the SQL Editor SpeedBar button, and specify the name of the file, or right-click the SQL Editor SpeedBar button and choose Open.
- To enter (and execute) a new SQL statement, choose File | New | SQL File or right-click the SQL Editor SpeedBar button and choose New.
- To view the SQL equivalent of an open QBE query, choose Query | Show SQL or click the SQL Editor SpeedBar button.

SQL-enabled ObjectPAL

SQL Link provides new ObjectPAL methods that support SQL and transaction processing. It also provides additional remote access capabilities for existing ObjectPAL methods. If you are familiar with SQL, you can also include SQL statements in ObjectPAL code using an **SQL . . . ENDSQL** block.

This chapter provides information on ObjectPAL methods compatible for use in an SQL environment.

For detailed information on ObjectPAL methods, see the Paradox for Windows NEWPAL.DB database.

ObjectPAL SQL methods

SQL Link adds a new SQL object type to ObjectPAL. Table 4.1 lists the methods for the SQL object type.

Table 4.1 ObjectPAL SQL methods for the SQL object type

New Method	Object Type	Description
executeSQL	SQL	Executes an SQL statement. A database must be specified.
executeSQLFile	SQL	Executes an SQL statement contained in a file. A database must be specified.
executeSQLString	SQL	Executes an SQL statement contained in a string. A database must be specified.
writeSQL	SQL	Writes the pass-through SQL statement (represented by the SQL variable) to the specified file.

Table 4.2 lists the syntax for the above methods.

Table 4.2 ObjectPAL methods for the SQL object type: syntax

Method Name	Syntax
executeSQL	<ol style="list-style-type: none"> 1. executeSQL(const <i>db</i> Database) Logical 2. executeSQL(const <i>db</i> Database, const <i>ansTbl</i> String) Logical 3. executeSQL(const <i>db</i> Database, const <i>ansTbl</i> Table) Logical 4. executeSQL(const <i>db</i> Database, var <i>ansTbl</i> TCursor) Logical
writeSQL	writeSQL (const <i>fileName</i> String) Logical
executeSQLFile	<ol style="list-style-type: none"> 1. executeSQLFile(const <i>db</i> Database, const <i>fileName</i> String) Logical 2. executeSQLFile(const <i>db</i> Database, const <i>fileName</i> String, const <i>ansTbl</i> String) Logical 3. executeSQLFile(const <i>db</i> Database, const <i>fileName</i> String, const <i>ansTbl</i> Table) Logical 4. executeSQLFile(const <i>db</i> Database, const <i>fileName</i> String, var <i>ansTbl</i> TCursor) Logical
executeSQLString	<ol style="list-style-type: none"> 1. executeSQLString(const <i>db</i> Database, const <i>sqlString</i> String) Logical 2. executeSQLString(const <i>db</i> Database, const <i>sqlString</i> String, const <i>ansTbl</i> String) Logical 3. executeSQLString(const <i>db</i> Database, const <i>sqlString</i> String, const <i>ansTbl</i> Table) Logical 4. executeSQLString(const <i>db</i> Database, const <i>sqlString</i> String, var <i>ansTbl</i> TCursor) Logical

Table 4.3 shows methods that were added to existing types to support SQL.

Table 4.3 SQL-enabled ObjectPAL methods for other object types

New Method	Object Type	Description
beginTransaction	Database	Starts a transaction on a server when transactions are supported. Otherwise, the function returns an error. Only one transaction for each database (or Alias) is allowed.
commitTransaction	Database	Commits all changes within a transaction on a server when transactions are supported. Otherwise, the function returns an error.
rollbackTransaction	Database	Rolls back all changes within a transaction on a server when transactions are supported. Otherwise, the function returns an error.
enumAliasLoginInfo	Session	Writes the Alias name, properties, and property values to a specified table.
forceRefresh	TCursor	Causes all data to be refreshed.
setBatchOn	TCursor	Turns on batch transaction processing.
setBatchOff	TCursor	Turns off batch transaction processing.
forceRefresh	UI Object	Causes all data to be refreshed.
getAliasProperty	Session	Returns the requested Alias property.
setAliasPassword	Session	Sets the Alias password.
setAliasProperty	Session	Sets the requested Alias property.

Table 4.3 SQL-enabled ObjectPAL methods for other object types

New Method	Object Type	Description
isOnSQLServer	TCursor	Returns True if the table is on an SQL server.
isOpenOnUniqueIndex	TCursor	Returns True if the TCursor is opened on a unique index. Frequently used for SQL server verification because many functions require TCursor opened on a unique index.

Table 4.4 lists the syntax for the above methods.

Table 4.4 SQL-enabled ObjectPAL methods for other object types: syntax

Method Name	Syntax
beginTransaction	beginTransaction ([const <i>isoLevel</i> String]) Logical
commitTransaction	commitTransaction () Logical
rollbackTransaction	rollbackTransaction () Logical
SQL . . . ENDSQL	SQL SQL <i>statements</i> ENDSQL
enumAliasLoginInfo	enumAliasLoginInfo (const <i>tableName</i> String, const <i>databaseName</i> String) Logical
getAliasProperty	getAliasProperty (const <i>aliasName</i> String, const <i>property</i> String) String
setAliasPassword	setAliasPassword (const <i>aliasName</i> String, const <i>password</i> String) Logical
setAliasProperty	setAliasProperty (const <i>aliasName</i> String, const <i>property</i> String) String
isOnSQLServer	isOnSQLServer () Logical
isOpenOnUniqueIndex	isOpenOnUniqueIndex () Logical

Standard ObjectPAL methods that support SQL Link

Table 4.5 lists the ObjectPAL methods and procedures that accept Aliases to SQL databases, but otherwise perform as they do in Paradox.

Table 4.5 Standard ObjectPAL methods that support SQL Link

Method	Object Type	Method	Object Type
open	Database	cSamStd	TCursor
isTable	Database	cSamVar	TCursor
executeQBE	Query	cStd	TCursor
executeQBFile	Query	cSum	TCursor
executeQBEStrng	Query	currRecord	TCursor
isAssigned	Query	cVar	TCursor
query (Keyword)	Query	deleteRecord	TCursor
writeQBE	Query	didFlyAway	TCursor
addAlias	Session	dropIndex	TCursor
enumAliasNames	Session	edit	TCursor
enumDatabaseTables	Session	empty	TCursor

Table 4.5 Standard ObjectPAL methods that support SQL Link

Method	Object Type	Method	Object Type
enumDriverCapabilities	Session	end	TCursor
enumDriverInfo	Session	endEdit	TCursor
enumDriverNames	Session	enumFieldNames	TCursor
enumDriverTopics	Session	enumFieldNamesinIndex	TCursor
enumEngineInfo	Session	enumFieldStruct	TCursor
enumOpenDatabases	Session	enumIndexStruct	TCursor
lock	Session	enumLocks	TCursor
removeAlias	Session	enumTableProperties	TCursor
setRetryPeriod	Session	eot	TCursor
unlock	Session	fieldName	TCursor
add	Table	fieldNo	TCursor
attach	Table	fieldSize	TCursor
cAverage	Table	fieldType	TCursor
cCount	Table	fieldValue	TCursor
cMax	Table	getLanguageDriver	TCursor
cMin	Table	getLanguageDriverDesc	TCursor
cNpv	Table	home	TCursor
cSamStd	Table	initRecord	TCursor
cSamVar	Table	insertAfterRecord	TCursor
cStd	Table	insertBeforeRecord	TCursor
cSum	Table	insertRecord	TCursor
cVar	Table	isAssigned	TCursor
empty	Table	isEdit	TCursor
enumFieldNames	Table	isEmpty	TCursor
enumFieldNamesinIndex	Table	isValid	TCursor
enumFieldStruct	Table	locate	TCursor
enumIndexStruct	Table	locateNext	TCursor
fieldName	Table	locateNextPattern	TCursor
fieldNo	Table	copyToArray	TCursor
fieldType	Table	locatePattern	TCursor
isAssigned	Table	locatePrior	TCursor
isEmpty	Table	locatePriorPattern	TCursor
isTable	Table	lock	TCursor
lock	Table	lockRecord	TCursor
nFields	Table	lockStatus	TCursor
nKeyFields	Table	moveToRecNo	TCursor
nRecords	Table	moveToRecord	TCursor
rename	Table	nextRecord	TCursor
setFilter	Table	nFields	TCursor
setIndex	Table	nKeyFields	TCursor
sort	Table	nRecords	TCursor
subtract	Table	open	TCursor
type	Table	postRecord	TCursor
unattach	Table	priorRecord	TCursor
unlock	TCursor	qLocate	TCursor

Table 4.5 Standard ObjectPAL methods that support SQL Link

Method	Object Type	Method	Object Type
add	TCursor	recNo	TCursor
atFirst	TCursor	recordStatus	TCursor
atLast	TCursor	seqNo	TCursor
attachToKeyViol	TCursor	setFieldValue	TCursor
bot	TCursor	setFilter	TCursor
cancelEdit	TCursor	setFlyAwayControl	TCursor
cAverage	TCursor	skip	TCursor
cCount	TCursor	sortTo	TCursor
close	TCursor	subtract	TCursor
cMax	TCursor	switchIndex	TCursor
cMin	TCursor	tableName	TCursor
cNpv	TCursor	type	TCursor
copy	TCursor	unlock	TCursor
copyFromArray	TCursor	unlockRecord	TCursor
copyRecord	TCursor	updateRecord	TCursor

Standard ObjectPAL methods that do not support SQL

Table 4.6 lists ObjectPAL methods and procedures that do not support SQL Link.

Table 4.6 ObjectPAL methods that do not work with SQL Link

Method	ObjectType	Method	ObjectType
compact	Table	enumRefIntStruct	TCursor
enumSecStruct	Table	enumSecStruct	TCursor
enumRefIntStruct	Table	familyRights	TCursor
familyRights	Table	fieldRights	TCursor
isEncrypted	Table	fieldUnits2	TCursor
isShared	Table	isEncrypted	TCursor
protect	Table	isRecordDeleted	TCursor
reIndex	Table	isShowDeletedOn	TCursor
reIndexAll	Table	isShared	TCursor
setExclusive	Table	protect	TCursor
setReadOnly	Table	reIndex	TCursor
showDeleted	Table	reIndexAll	TCursor
tableRights	Table	showDeleted	TCursor
unprotect	Table	tableRights	TCursor
usesIndexes	Table	undeleteRecord	TCursor
compact	TCursor	unprotect	TCursor

Using SQL Link: advanced topics

This chapter introduces some advanced SQL Link concepts. It includes a description of table properties that have particular control over SQL Link capabilities and behavior, and how SQL Link handles transactions and record locking.

For a discussion of how to migrate existing ObjectPAL applications to an SQL environment or develop SQL applications using ObjectPAL, see Chapter 6.

Table requirements for use of SQL Link capabilities

There are two table properties which have particular control over Borland SQL Link capabilities and behavior: a unique row identification method, and a defined row ordering.

Unique row identification

A unique index guarantees unique row identification, but some SQL servers can also identify individual rows in the absence of an explicit unique index. This server mechanism does not apply to all server objects. For example, even if your server supports an implicit unique row identification method for tables, it may not support one for SQL server views. (Your server-specific *Connecting To...* manual notes whether your table supports an implicit unique row identification method.)

Borland SQL Link requires a unique row identification method in order to enable some editing capabilities, and to provide full blob access support for some SQL servers.

Enabling editing capabilities

The editing capabilities that require unique row identification are QBE DELETE and CHANGETO queries that are performed on the client (for example, heterogeneous queries) and individual record modifications or deletions made on a form or in ObjectPAL.

Note QBE INSERT queries, SQL Editor queries, and insertions made in ObjectPAL or on a form, do not require uniquely identifiable rows.

Blob access support

For SQL servers that do not support blob handles for random reads and writes, full blob support requires uniquely identifiable rows. Otherwise SQL Link sequentially reads as much of the blob as the server allows when first reading the row, then saves the contents in memory. If, after several more reads, the contents are not requested, the memory is freed. The blob is no longer accessible.

Most SQL servers limit a single sequential blob read to less than the maximum size of a blob. In those cases an entire blob may not be available. QBE, forms, and ObjectPAL do not support inserting and altering sequentially-accessed blobs. To see if your SQL server tables support blob handles, or identify the maximum size of a single blob read, see Chapter 3 of your server-specific *Connecting to...* manual.

Defined row ordering

An index is required to define a row ordering. If there is no index and the SQL server provides a method to identify individual rows, SQL Link can also use this method to define a row ordering. However, an index is preferred by SQL Link, because using an index for ordering almost always provides better performance.

SQL Link requires a defined row ordering to display a small window into the server data. This allows for quick access to data at the client, and provides the ability to refresh that window with the latest data. The window into the server data moves as the client current row location moves, and can be refreshed from the server through ObjectPAL or by using *Ctrl+F3*.

If no defined row ordering is available, SQL Link saves all information read (not just a small window of data), and cannot provide the ObjectPAL or the interactive user refresh operation. For example, if SQL Link is asked to move to the end of a data set with no defined row ordering, the entire data set is saved. Although memory usage is limited to the configuration values, the client must have enough disk space to save this data.

Insertions can be supported against SQL server objects that do not have defined row orderings, but SQL Link cannot tell where the SQL server places the inserted row. Therefore, the row may or may not appear in the set of data as it is read. Different SQL servers have different mechanisms for inserting rows, so this behavior can vary from SQL server to SQL server, and even from table to table. Of course, if a row cannot be inserted for any reason, SQL Link reports the problem.

Locking and transaction support

SQL Link provides direct support for both table and record locking, and explicit SQL transaction management. Pass-through SQL may also be used to control SQL locking and transactions. Pass-through SQL is treated as a different SQL server session, and must be fully controlled by the client. SQL Link completely adopts the semantics of the SQL server for pass-through SQL. Because the locking and transaction behavior of SQL

servers varies greatly from vendor to vendor, see your SQL server documentation for information on its specific behavior.

Default transaction behavior

SQL operations always take place within the context of a transaction. When no explicit transaction occurs, SQL Link manages the SQL server transactions transparently for the client. Any successful modification of SQL server data is immediately committed to ensure its permanence in the database. For example, a single QBE query, a single ObjectPAL table append, a single form edit operation, and a single ObjectPAL record post or unlock are each individually committed by default.

Even if multiple operations take place (for example, in a heterogeneous QBE query), they appear to the client as a single operation. This property is called *statement atomicity*: either the whole operation is done, or none of it is done. The client has no clean-up work to do if only part of the operation completes.

Table-locking behavior

Since table locking is a familiar feature to Paradox users, SQL Link provides a degree of support for table locking if the SQL server supports it. Different SQL servers provide different levels of support for table locking. Some servers provide no table locking support at all. Others only provide support for read-only locking (many clients can share a lock and all can read). Some SQL servers provide support for locking, but require the client to wait until a lock is granted, rather than letting the client know immediately if the lock could not be achieved. For information on locking support provided by your SQL server, see your server documentation.

SQL servers that support table locks maintain a lock within the context of a transaction: a lock can only be acquired within a transaction, and only released by terminating the transaction. This is sometimes referred to as a *two-phase locking* protocol. When SQL Link is asked to acquire a table lock, it automatically starts a transaction if necessary. When asked to release a table lock, SQL Link must commit the transaction in order to release the lock. Because a transaction commit releases all locks, SQL Link automatically re-acquires any remaining locks.

Note If a table lock is held when a commit becomes necessary, a time window exists in which the lock is not held and unanticipated changes can occur. For this reason, it is recommended that all table locks be released together when the last lock is needed, or that explicit SQL transactions be used instead of table locking.

Record-locking behavior

SQL servers automatically and transparently lock data as required, although different SQL servers vary in the type of lock used, and how granular the lock is. For example, some servers provide individual record locks, while others can only lock a group, or page, of records. Also, some servers provide automatic record versioning or database snapshots so that other copies of data being modified can be read by clients instead of waiting for a modification to finish.

In addition to the automatic locking that SQL servers provide, Borland SQL Link provides a particular type of record locking called *optimistic locking*. Optimistic locking allows a client to make changes to a local copy of the record without the performance and concurrency penalty incurred by asking the server for a lock over the modification duration. When the client modifications are finished, the current SQL server record is first checked to make sure no changes have occurred to the record, then the modifications are completed. The operation is said to be optimistic because it assumes that no other client will change the record, but then makes sure of that as the final change is sent to the SQL server.

If the record was changed, an optimistic lock failure occurs. The client is notified that the requested operation cannot be performed because someone else has changed the data. The client can then inspect the new data and decide whether or not to make changes at that time.

Because server data cached on the client can immediately become out of date at the server, SQL Link always performs optimistic locking. This protects the client against inadvertently changing data that has never been inspected.

Client-controlled transaction behavior

SQL Link provides ObjectPAL methods to explicitly start, commit, and roll back a transaction. When a transaction is explicitly started, a COMMIT operation is never performed automatically, so the client has the full protection of the SQL server transaction mechanism. That is, modifications are performed, but not automatically committed and table lock releases do not cause a commit within an explicit client transaction. Once a transaction has been started, the client can use ObjectPAL to ask if a transaction is in effect. When starting a transaction, SQL Link uses the default isolation level for the SQL server, but different SQL servers have different defaults.

An SQL Link transaction is always started and ended through a database Alias. Once started, all modification, table, and record locking operations executed in that database (other than pass-through SQL) are performed in the context of the started transaction. Therefore, a commit will make all operations since the start of the transaction permanent in the SQL server database. A roll back will undo all operations.

When operating within an explicit client transaction, some operations performed by SQL Link cannot be made atomic. QBE queries executed on the client, and table append operations execute in their own transaction by default. When executed within an explicit client transaction, they execute within the client transaction context. If such an operation fails during processing, the operation may be only partially complete. An ObjectPAL developer may wish to roll back the transaction when an error is returned from one of these operations.

Although transactions can be started in multiple database Aliases simultaneously, SQL Link manages the transactions independently, so the client must be careful to minimize the window of vulnerability after the first commit of a multiple database transaction. A failure after the first commit but before subsequent commits will cause committed data to remain permanent, and uncommitted data to be undone.

Note *Data Definition Language* (DDL) operations can be done within an explicit transaction if the SQL server allows it. Most SQL servers do not allow this, or would implicitly commit the explicit transaction. For such servers, SQL Link disallows DDL operations when an explicit client transaction is in effect.

Transaction and locking errors for ObjectPAL programmers

New errors exist to indicate deadlock errors and optimistic lock failures:

peDeadlock

This error indicates that the SQL server has detected a deadlock condition. If the SQL server rolls back transactions upon deadlock, any explicit client transaction is effectively rolled back. Otherwise, the client may choose an alternate action, or roll back the transaction explicitly. Due to the automatic lock acquisition mechanism that SQL servers use, a deadlock error can occur during any operation; typically during a client request to insert, delete, or modify records.

peOptRecLockFail

This error indicates that SQL Link has identified an optimistic lock failure. The record contents have changed, but the record still exists. It can be retrieved and re-examined as necessary.

peOptRecLockRecDel

This error indicates that SQL Link has identified an optimistic lock failure, and the record has apparently been deleted. The record can be re-inserted, or an alternative action can be taken.

Developing ObjectPAL applications

This chapter discusses strategies for ObjectPAL developers who want to migrate to the SQL environment. It includes information on how to move Paradox or dBASE tables to an SQL server, re-bind an existing form to an SQL server, and work with the SQL Link locking strategy.

For an introduction to the ObjectPAL programming language, see the Paradox manual *Learning ObjectPAL*. For an intermediate-to-advanced guide to ObjectPAL, see the *Paradox ObjectPAL Developer's Guide*.

For information on advanced SQL Link concepts, see Chapter 5.

Moving a local application to an SQL environment

This section discusses tasks that must be completed to port a local ObjectPAL application to an SQL environment. The procedures in this section are designed to get an existing ObjectPAL application up and running quickly against an SQL server.

The basic procedures discussed are:

- Uploading local tables to the SQL server
- Re-binding existing application forms
- Changing table references
- Changing index references

For a discussion of ways to optimize code that has been ported to an SQL environment, see the following section.

Moving local tables

Before you begin to move your application to the server, upload all the tables you want to reside on the server in either of the following ways:

- Use the SQL Editor to create the table on the server. When the table is ready, use a QBE INSERT query to copy the desired records from the local table to the server table.
- Write an ObjectPAL script.

Example 6.1 Moving local tables to an SQL server

In this example, the tables to be uploaded all reside in C:\PDOX\DATA. The target is an InterBase server whose Alias is IB_DATA.

```
var
  tb Table
  fs FileSystem
  tblName String
endvar
while fs.findNext( "c:\\pdx\\data\\*.db" )
  tblName = fs.name()
  tb.attach( fs.fullName() )
  ; strip the table extension ".db"
  tb.copy( ":IB_data:" + substr( tblName ,1, search( tblName , "." )-1) )
endwhile
```

After all local tables are uploaded to the SQL server, index the tables to duplicate the links between tables that exist in your local application. You can use either SQL statements or the ObjectPAL INDEX statement to create the index. (For the syntax of the INDEX keyword, see the *ObjectPAL Reference*.)

To ease migration, preserve the table names (without the file extension) and the index names whenever possible. If you have two or more local tables that use the same index name and your server only allows you to use an index name once in a database, you may need to alter the index name. If so, you also need to change the existing references to the changed index.

Re-binding forms

Re-binding an existing form to use SQL server tables requires that all local tables be uploaded to the SQL server, that the SQL database have a valid Alias, and all necessary indexes have been created. In addition, any forms which use SQL tables must point specifically to tables that reside on the aliased server.

If you originally created forms by specifying an Alias explicitly, you may only need to change the Alias so that it maps to the SQL server. The forms can then be used in the new application. However, if you want to use forms that were created using tables in your working directory or some hard-coded DOS path name, you must re-map the form to the SQL server Alias.

You can do this in either of the following ways:

- Move the tables in the form to a different location on the workstation hard disk, then load the form in Design mode. This enables you to manually re-associate the form's supporting tables. Select the appropriate server table; Paradox re-binds the data model to the selected tables.

For detail tables in a data model, if Paradox cannot find the index named in the link, it attempts to find and use an index which corresponds to the link information stored in the data model. If the table is a master and the index is not found, Paradox will also attempt to open the correct index; however, due to the changes in the index, you will lose any Order/Range information stored with your form.

- Enter the data model of the form. Re-construct the data model directly so that it uses the desired Alias:
 - 1 Use the Alias Manager to create an Alias which points to the directory where the tables reside.
 - 2 Open the form in design mode.
 - 3 Enter the data model diagram.
 - 4 Unlink and remove the tables in the data model.
 - 5 Recreate the data model. Be sure to select an Alias from the Path drop-down list. After you select an Alias, select the table. Paradox references the table by the Alias and stores it in the data model with that Alias.
 - 6 Re-link the tables in the data model.
 - 7 Save and close the form.

Changing table references

As with forms that were originally created for local use, any tables used in an SQL environment must point specifically to tables that reside on the aliased server.

Because Paradox uses Aliases to make connections with SQL databases, applications which were originally written against absolute DOS directories must be modified to run against an SQL server. For example, you may need to change all occurrences of:

```
Var
    tcCustomer TCursor
endvar
tcCustomer.open( "C:\\data\\Customer.db" )
```

to:

```
Var
    tcCustomer TCursor
endvar
tcCustomer.open( ":someAlias:Customer" )
```

where `:SomeAlias:` is an Alias which points to `C:\DATA`. If your local application already uses Aliases in this manner, moving the application to the server is transparent. You may redefine `:SomeAlias:` to point to your server database, and put `CUSTOMER` on the server.

The TCursor opens the SQL table instead of the local table, and almost all TCursor methods called work properly against the server.

The use of Aliases makes it easier to port an application to a different server, since only the Alias definition (and not the code itself) must change.

Changing index references

As mentioned above, SQL servers have different naming conventions for index names. If you had to change index names in order to prevent duplicate index names from existing on the server, look for occurrences of the **SwitchIndex()** method. Change those references so that they point to the new index name at the SQL server.

Note This is especially true of Paradox primary keys that are uploaded to an SQL server. In Paradox the primary key (the default index with which a Paradox table is opened) is not given a name. Since SQL requires names for all indexes, it may be necessary for you to change occurrences of **SwitchIndex()** to **SwitchIndex(<SQLKeyIndexName>)**.

After all forms have been remapped and all table and index references changed to point to the SQL server, check the application for the presence of ObjectPAL methods that do not work against SQL (see Chapter 4, Table 4.6). Remove these methods from the application, or change them to SQL equivalents.

Using transactions in ObjectPAL applications

This section discusses the SQL locking strategy, and how transaction management can be brought into an ObjectPAL application.

The decision to use transactions in an ObjectPAL application is not clear-cut. It depends upon the update requirements of your application, the number of users expected to access data concurrently, and even the desired level of performance in your application, since interactive editing against a server will be somewhat slower than against local tables.

The SQL locking strategy

SQL servers do not use the same locking protocols as non-SQL desktop applications. SQL Link's optimistic locking scheme (see Chapter 5) presents unique challenges to the application developer. Although your application may now be up and running on an SQL server, you may need to resolve issues with the way your application code manages locks in order to have the best possible SQL application.

For example, your application may be an order entry database written to run under Paradox. The application:

- Enables the user to define or locate a customer in the database (and locks the master record).
- Generates a unique order number.

- Enables the user to enter detail line items for this order and posts each line item as it is entered. Since the master record is locked, the line items are also protected.
- Unlocks the order record.

This approach works because the master record is locked whenever anyone attempts to modify it. In SQL, however, no such guarantee exists. Another user can edit the first user's order record while the first user is still working with it. If the other user deleted the record before the first user finished, the first user's entry would end in error.

SQL offers a solution to this problem by means of transactions (see Chapter 1). SQL-enabled ObjectPAL uses the transaction capabilities of SQL Link to begin, commit, and roll back transactions. This guarantees a consistent set of changes. The SQL server also automatically places locks whenever a user accesses and updates data on the server. These automatic locks prevent other users from modifying the changes you have posted before the transaction is committed or rolled back.

Applying transaction management principles

The following example shows how transactions can be used in ObjectPAL applications.

Example 6.2 Using transactions in an ObjectPAL application

In the example situation, you want to modify the previously-described order entry application so that a unique order number is created whenever the user requests that a new order be created. You do not have a stored procedure or trigger on the server to automatically generate the unique order number, and need to generate it from within your ObjectPAL application.

One technique, used often against Paradox tables, is to have a table with one field that contains the next available number for use as an order number. To automatically generate a unique order under, your program needs to read the number, increment the number, and prevent any other user from taking the same order number.

In non-SQL-enabled ObjectPAL, explicit record locking accomplishes this goal. In an SQL environment, however, the solution is to place the unique number generation inside a transaction.

The following code might go in the Action method of the order record object on the form:

```
var
    db Database
    tcNumber TCursor
    i Smallint
endvar
db.open( ":IB_Data:" )
if eventInfo.id() = dataInsertRecord Then

; Allow only one transaction for each order. db is a Database var pointing to your
; server database.

    if db.transactionActive() then
        msgStop( "Order in progress", "Please finish the current orderfirst")
```

```

    disableDefault                ; refuse the insert
    return
endif

Orders.edit()                    ; Transaction - generate a unique number.

db.beginTransaction()
tcNumber.open( ":IB_Data:OrderNo" ) ; Numbers table on Interbase
tcNumber.edit()
i = 0
Try
    newOrder = tcNumber.(1)        ; one field exists: the next available number.
    ErrorTrapOnWarnings( Yes )    ; guarantee that we trap anything that goes wrong.
    tcNumber.(1) = newOrder+1     ; this fails if someone else is updating.
    tcNumber.unlockRecord()       ; attempt to commit the record.
OnFail
    if i > 10 then
        ; // time out.
        db.rollbackTransaction() ; cancel the transaction.
        tcNumber.currRecord()    ; reset the tcursor buffer data
        ErrorTrapOnWarnings( No ) ; restore warning level
        disableDefault           ; don't insert a record.
        return                   ; Quit.
    endif
    i = i + 1
    Retry                        ; Try again.
EndTry
ErrorTrapOnWarnings( No )
tcNumber.close()
db.CommitTransaction()          ; Release the Numbers table and we're done.
Endif

```

This code ensures that if two people read the value and one updates it, the second user receives an error when attempting to update the record. The programmer can then trap for a concurrency problem to see if someone else has modified the data while the first user was at work.

If you are not using a transaction to protect the record that has been locked and someone else deletes the record, you might lose the lock. In this case, the next time Paradox attempts to paint the data on your display you receive an Action method ID: **DataLostLock**. Paradox then attempts to drop your lock and sends you an Action method with the ID **DataCancelRecord**.

Note One restriction to locking and unlocking records in transactions is that a lock/unlock sequence for any given record must either occur entirely inside or outside a transaction block. This means a record may not be locked outside of a transaction and then unlocked within the transaction. The record also may not be locked inside a transaction and unlocked outside the transaction. Either of the previous situations return a Transaction Mismatch error. If a record is locked and then changed in the course of a transaction, unlocking the record does not post the changes to the server table. If the TCursor goes out of scope, the changes are lost.

“Snapshot” transactions

SQL Link supports only one open ObjectPAL or SQL transaction per database. This means that once a user begins a transaction against a database, everything the user does interactively will occur inside that transaction until the transaction is either committed or rolled back. While this provides a good amount of concurrency protection on a multi-table form, it has the disadvantage of tying up the records being accessed until the transaction is completed. In an order entry application where a user might take an indefinite period of time to complete an order, it may be inadvisable to place a transaction around an entire order. Instead, you might consider using the 'snapshot' method of application development.

In a program using snapshot transactions, when the user requests SQL data the server downloads the data into local Paradox or dBASE tables. After the user updates the data locally, the SQL database is updated in a transaction that occurs after the user has finished the update. The server tables are updated all at the same time using either an SQL query or a TCursor-based transaction.

Tip For some queries, you can design a form which uses a QBE file in its data model which runs against the server. The form should provide local editing capabilities against the resulting *Answer* table. A slight modification to this same QBE allows users to post updates back up against the server, with little coding involved.

An advantage of the snapshot method is that server data is not locked for long periods of time, and still gains the benefits of transaction processing during the update. A disadvantage is that this method requires more programming work to correctly detect errors, pinpoint their cause, and resolve the situation.

Note Querying all data to local tables and performing edits may cause data loss due to the conversion of SQL data types to their Paradox or dBASE equivalents. (For information on data types supported by your SQL server, see Chapter 3 of your server-specific *Connecting to...* manual.)

Generating the key at the end of the transaction

Another solution is to generate unique values when posting the data, instead of generating them at the time the record is created. This creates fewer data locks while transactions are in progress, and thus less contention for data. The code for posting the data can be very similar to the code in Example 6-2, except that the posting process can be broken into two transactions:

- 1 Obtain the unique ID, increment it, and commit the change.
- 2 Put the number into the record, post it, and commit the number.

In this method, concurrency problems can only be resolved at the time the user is finished with the record. Editing sessions may run for a long period of time, or even be cancelled, without causing lock conflicts.

Optimizing your SQL application

This section introduces strategies to be considered when optimizing your ObjectPAL application for use in an SQL environment.

TCursors vs. queries in an application

If you port into an SQL environment a Paradox application which makes extensive use of TCursors, you will see that TCursors behave exactly as they do within Paradox. In one sense this is a great benefit, since it enables you to run code unmodified against either server or local data. However, many programs use filters and scan loops to modify sets of data, and would benefit more from the use of SQL queries.

The following example illustrates how to use a TCursor in a program that calculates the total of several items in a line item set for the cursor order. The next example illustrates that using a query can obtain similar results.

Example 6.3 Using a TCursor to calculate the total of line items in a set

```
var
    tc TCursor
endvar
tc.attach( LineItemDetail ) ; where LineItemDetail is a
                           ; tableframe for the current order.

tc.edit()
Scan tc :
    tc."Total" = tc."Quantity" * tc."Price" * 1.085
endScan
tc.endEdit()
```

This code performs even faster than a query against local Paradox data, because there is already a cursor to a subset of the Line Items table using a known index. The code scans through and updates values quickly in this filtered set. However, this may not be a fast operation against a remote SQL server.

When the above code executes, SQL Link cannot distinguish that the code is performing an operation against a set. Therefore, it is unable to optimize the operation at the SQL server. SQL Link must download the SQL data to Paradox (which performs the calculation) then issue an UPDATE SQL query to update that record. For one or two records, the difference in speed does not matter, but as the number of processed records rises the operation becomes less efficient.

Since one of the primary advantages of SQL is its ability to issue queries against large amounts of data and decrease network traffic, TCursor implementations can put SQL at a disadvantage. In this case, an SQL query might create a more optimized transaction.

Example 6.4 Calculating the total of line items in a set using a query

```
Query
LineItems | OrderNo      | Quantity | Price | Total |
          | ~(OrderNo.value) | _quant  | _price | changeto_quant*_price*1.08|
EndQuery
```

If you save this query as CALCPRICE.QBE, you may execute it from within your code:

```
var
  db Database
endvar
db.open( ":IB_Data:" )
executeQBEFile( db, "Calcprice" )
```

The above method takes full advantage of SQL server query optimization and decreases network traffic. Much of the heavy network traffic which occurs in networked Paradox applications can be eliminated when this approach is used effectively.

Caution If you send the SQL statement to the server using the SQL Editor window, be aware that the issued query is considered part of a different connection than QBE, ObjectPAL, or a form, and thus cannot participate in an open ObjectPAL transaction. If you wish the query to participate in an open transaction, frame the query in the QBE environment instead of passing an SQL string directly to the server.

Lookup tables

Unlike Paradox tables, SQL tables do not provide a built-in lookup help and fill feature, which is used heavily in Paradox applications. You need to create a TCursor against the SQL table and provide explicit code to look up the entered value manually. If your local ObjectPAL application already provides some type of lookup capability, your application can run against an SQL server without change.

When there are many small lookup tables that can be easily relocated to local tables, you might consider using them locally. This will increase the performance of your applications, particularly if you are performing interactive transactions against the server and you expect the lookup to be performed many times. Since doing a TCursor lookup against a SQL table involves a query, moving the table to a local directory speeds up the process significantly.

It is important to remember that SQL Link uses TCursors to provide transparent access to both SQL data and Paradox data. This enables you to choose which tables to upload to the server and which tables you wish to be on the network or local hard drive, in order to obtain the best possible performance in a busy network environment.

Stored procedures and triggers

Many kinds of data validation and unique key generation can be performed at the SQL server by triggers or stored procedures. These can decrease the amount of application code you write at the client and improve the overall performance of the application. Whenever you plan to port code that should occur when the user inserts, deletes, or updates a record, consider whether the operation could easily be performed by a trigger or stored procedure.

Index

Symbols

% operator 23
.. operator 23
@ operator 23
_ operator 23

A

accessing SQL data
 embedding SQL statements 1
 using Database Desktop 1
 using Paradox 1
 writing ObjectPAL applications 1
Alias 12, 14
Alias Manager 9, 10
Aliases 41, 42
 using to connect with SQL databases 41
alphabetic data 14
applications
 moving to SQL environment 39

B

beginTransaction
 defined 28
 syntax 29
blobs 34
 handles 34
 sequential reads 34
Borland desktop product
 SQL-enabled state 1
Borland SQL Link documentation
 how to use 2
Borland SQL Link User's Guide
 audience 1
 how to use 1
 where to find information 3
buttons
 Data Model 13
 SQL 15

C

case-sensitive search 24
CHANGED.DB table 14
CHANGETO 25, 33
character set
 desktop product 14
 SQL server 14
character sets 14

Check 22, 24
Check+ 23
client 6
client applications 6
client workstation
 requirements 11
client/server environment 5, 7
 SQL Link in 9
client/server model 7
client/server system 6
code pages 14
column
 definition of 15
columns
 retrieving specific 20
COMMIT 7, 8
commitTransaction
 defined 28
 syntax 29
conventions 3
 ObjectPAL syntax 4
 sorting 25
 typefaces used 3
 uppercasing 25
converting SQL data types 45
Ctrl+F3 12, 34

D

data
 retrieving all 19
data dictionaries
 as a feature of database servers 1
data locking 5
data model 41
Data Model button 13
data types 45
Database Desktop 1
 functions with SQL Link 9
 SQL-enabled 11
database server 6
 client 6
 communicating with 8
 description 6
 differences from network server 6
 query processing on 6
 querying directly 6
 similarities to network server 6
databases
 NEWPAL.DB 27

DataCancelRecord 44
DataLostLock 44
Date, C. J. 7
DBD.EXE 1, 9
defined row ordering 34
 when unavailable 34
DELETE 7, 18, 25, 33
DELETED.DB table 14
DESC 24
desktop product
 differences when SQL-enabled 11
 operation when SQL-enabled 12
 working directories 12
desktop product queries
 SQL equivalents 18
desktop product window 12
 when SQL-enabled 12
detail tables 41
 data model in 41
developing ObjectPAL applications 39
DISTINCT 22, 23
duplicate entries
 eliminating 22

E

enumAliasLoginInfo
 defined 28
 syntax 29
environment
 client/server 5, 7
 network 5
environment
 SQL 5
ERRCHG.DB table 14
ERRDEL.DB table 14
ERRINS.DB table 14
executeSQL
 defined 27
 syntax 28
executeSQLFile
 defined 27
 syntax 28
executeSQLString
 defined 27
 syntax 28
explicit record locking 43
extended character sets 14

F

F6 23

field

definition of 15

forceRefresh

defined 28

Form I Data Model 13

Form windows

in an SQL environment 13

forms

general changes in 13

re-binding during migration 40

remapping during migration 40

using with QBE files in data

model 45

G

gateway 5

generating the key at the end of

the transaction 45

getAliasProperty

defined 28

syntax 29

H

How to use this book 1

I

INDEX 40

index name 40

altering during migration 40

indexes

changing references 42

changing references to 40

unique 33

INSERT 7, 18, 25

INSERTED.DB table 14

international language drivers 14

isOnSQLServer

defined 29

syntax 29

isOpenOnUniqueIndex

defined 29

syntax 29

K

keywords

DESC 24

DISTINCT 22, 23

ORDER BY 24

L

LAN 5

language drivers 25

international 14

role in sorting 14

why necessary 14

Learning ObjectPAL 39

local area network 5

local folder

using with SQL database 12

local processing

queries 24

locking 5

explicit 43

optimistic 42

records 34

tables 34

lookup tables 47

M

master table 41

moving

applications to SQL

environment 39

N

network environment

using network server 5

network model 6

network server 5

access by more than one user 5

accessing data stored on 5

NEWPAL.DB 27

NOT 22

O

ObjectPAL

new SQL methods 27

transaction and locking errors 37

using transactions in 42

ObjectPAL applications

developing 39

using with SQL Link 10

ObjectPAL SQL methods 27, 28

ObjectPAL SQL methods

syntax 28, 29

ObjectPAL syntax 4

ODAPI Configuration Utility 25

ODAPI files

location of 12

ODAPI.CFG

specifying default settings in 12

operations

QBE query

eliminating duplicate

entries 22

pattern matching 23

retrieving all data 19

retrieving specific columns 20

retrieving specific rows 20

specifying alternate search

conditions 21

specifying multiple search

conditions 21

specifying negative search

conditions 22

specifying row order 24

operators

.. 23

@ 23

__ 23

Check 22

Check+ 23

NOT 22

optimizing SQL applications 46

ORDER BY 24

Order/Range 13, 41

ordering rows 34

P

Paradox for Windows

functions with SQL Link 10

Paradox forms 10

Paradox ObjectPAL Developer's

Guide 39

Paradox reports 10

pass-through SQL 10, 19

using 26

peDeadlock 37

peOptRecLockDel 37

peOptRecLockFail 37

preparing to use SQL Link 11

procedures

stored 47

Q

QBE 9, 10

description 17

in an SQL environment 14

typical operations 19

using in an SQL environment 17

using with SQL Editor 17

QBE query operations

eliminating duplicate entries 22

pattern matching 23

retrieving all data 19

retrieving specific columns 20

retrieving specific rows 20

- specifying alternate search conditions 21
- specifying multiple search conditions 21
- specifying negative search conditions 22
- specifying row order 24
- QPW.EXE 1, 9
- queries
 - CHANGETO 33
 - processing under QBE 9
 - to SQL databases 9
 - updating 25
- queries in applications 46
- query 6
 - processing 6
- query types
 - local processing defined 24
 - remote processing defined 24
- query window 10
- querying data
 - SQL tables 18
- querying SQL data 17

R

- RDBMS 7
- re-binding forms 40
- record locking 13
- record numbers 13
- records
 - locking 34
- references
 - changing index 42
 - changing table 41
- refreshing data displays
 - Ctrl+F3 12
 - in an SQL environment 12
- relational database 7
- relational database management systems 7
- remote processing
 - queries 24
- ROLLBACK 7, 8
- rollbackTransaction**
 - defined 28
 - syntax 29
- row
 - definition of 15
 - unique identification of 33
- row order
 - specifying 24
- row ordering
 - defined 12, 34
- rows

- accessing blobs 34
- adding 25
- ordering 34
- removing 26
- retrieving specific 20

S

- search conditions
 - specifying alternate 21
 - specifying multiple 20, 21
 - specifying negative 22
- searches
 - case-sensitive 24
- SELECT 7, 18
- SEQUEL 7
- sequential blob reads
 - limitations 34
- server 5
- setAliasPassword**
 - defined 28
 - syntax 29
- setAliasProperty**
 - defined 28
 - syntax 29
- Shift+F6* 23
- snapshot transactions 45
- sort order 14
- sorting
 - conventions 25
- SQL
 - advantages as tool 7
 - defined 7
 - history of 7
 - methods for ObjectPAL 12
 - pass-through 10, 19
 - pronunciation of 7
 - uses of 7
- SQL . . . ENDSQL**
 - syntax 29
- SQL applications
 - optimizing 46
- SQL button
 - purpose 17
 - pushing with no active query 18
- SQL data
 - converting data types 45
 - querying 17
- SQL data types
 - SQL server support for 45
- SQL database
 - connecting to 41
 - requirements to access 11
- SQL Editor 10, 13, 15, 17, 40
 - as query processing tool 15
- entering, executing, saving statements in 18
- viewing statements in 15
- viewing the SQL translation 18
- SQL Editor window 26
 - how to open 26
- SQL environment 5
 - migrating to 39
- SQL equivalents
 - desktop product queries 18
- SQL hourglass 12
- SQL keyword 27
- SQL Link
 - accessing SQL servers 9
 - advanced topics 33
 - description 8
 - in client/server environment 9
 - new capabilities with 12
 - new features 11
 - non-supporting ObjectPAL methods 31
 - preparing to use 11
 - record locking in 13
 - remote access features 27
 - summary of installation process 11
 - table requirements for 33
 - using Paradox with 8
 - using Quattro Pro with 8
 - writing ObjectPAL applications for 8
- SQL Link files
 - location of 12
- SQL Link installation
 - requirements for InterBase users 12
- SQL locking strategy 42
- SQL server
 - creating Alias for 12
 - moving local tables to 40
- SQL server software
 - prerequisites 11
- SQL servers
 - support for Date/Time fields 13
- SQL statements
 - COMMIT 7, 8
 - DELETE 7, 18
 - INSERT 7, 18
 - ROLLBACK 7, 8
 - SELECT 7, 18
 - UPDATE 18
- SQL table data
 - changing 13
 - updating through SQL Editor 13
- SQL tables
 - adding rows 25

- changing values 25
- in forms 13
- record locking in 13
- removing rows 26
- using forms to modify 13
- SQL terminology 15
- column 15
- row 15
- table 15
- SQL translation
 - viewing 18
- SQL-enabled ObjectPAL 27
- SQLQRYMODE 14, 25
- standard ObjectPAL methods
 - non-supporting SQL Link 31
 - supporting SQL Link 29
- statements
 - CHANGETO 25
 - DELETE 25
 - INSERT 25
- stored procedures 47
 - as a feature of database servers 1
- syntax
 - ObjectPAL SQL methods 28, 29
- syntax checking 26

T

- table
 - definition of 15
 - table names 40
 - table properties
 - .TVS files 13
 - functionality of 13
 - table windows
 - displaying record numbers in 13
 - for SQL data 13
 - general changes in 13
- tables
 - CHANGED.DB 14
 - changing references 41
 - DELETED.DB 14
 - ERRCHG.DB 14
 - ERRDEL.DB 14
 - ERRINS.DB 14
 - INSERTED.DB 14
 - locking 34
 - lookup 47
 - opening with TCursor 42
 - preserving names during migration 40
 - updating views of 12
 - uploading local 40
- TCursor 42, 44, 47
 - disadvantages of 46

- example of using 46
- TCursor-based transaction 45
- TCursors in applications 46
- terms and conventions 3
- thumb 13
- Timestamp 13
- Transaction Mismatch 44
- transaction processing 8
- transactions 34
 - defined 8
 - ending 8
 - example ATM transaction 8
 - example of in ObjectPAL 43
 - generating the key at the end of 45
 - in ObjectPAL applications 42
 - locking and unlocking records in 44
 - management principles of 43
 - ObjectPAL example 43
 - open 45
 - restrictions to 44
 - snapshot 45
 - TCursor based 45
 - transaction processing 8
- triggers 47
 - as a feature of database servers 1
- typical QBE query operations 19

U

- unique indexes 33
- UPDATE 7, 18, 46
- updating SQL database using QBE 25
- uploading local tables 40
- uppercasing
 - conventions 25
- uppercasing conventions 14
- using Form windows 13
- Using pass-through SQL 19
- using QBE 14
 - SQL database updates 25
- using Table windows
 - when SQL-enabled 13
- Using transactions in an ObjectPAL application 43

V

- values
 - changing 25
- viewing data
 - SQL tables 18

W

- WAN 5
- Where to find information 3
- wide area network 5
- working directories 12
 - in an SQL environment 12
- working in an SQL environment
 - SQL environment
 - working in 12
- working with international language drivers 14
- writeSQL**
 - defined 27
 - syntax 28

Y

- your SQL-enabled desktop product 11

Borland

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 431-1000. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part # SQG1110WW21770 • BOR 6059