PARADOX® SQL LINK

USER'S GUIDE

BORLAND

Paradox SQL Link

Version 4.0

User's Guide

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

10 9 8 7 6 5 4 3 R2

CONTENTS

TABLES and FIGURES

CHAPTER 1

Introduction

Paradox SQL Link lets you access data stored on SQL database servers using the familiar Paradox menus and the power of the Paradox Application Language (PAL).

Paradox 4.0 and SQL Link work together so that you can work with data stored on many database servers. SQL Link lets you work as you have in the past, using Paradox interactive menus and specialized PAL programs, and also lets you access remote SQL tables. You now have the power of SQL at your fingertips, without needing to learn SQL programming.

Structured Query Language (abbreviated SQL and commonly pronounced *sequel*) is the standard language for storing and manipulating data in *relational databases*. SQL database servers run on local area network (LAN) file-server systems, minicomputers, and mainframes. SQL database servers are often installed on dedicated, powerful machines that store and process large amounts of data quickly. A database server not only provides file sharing but also serves as an intelligent "back end" for user applications. Security constraints, data integrity rules, concurrency, and transaction processing are all handled by the database server.

Paradox SQL Link serves as the "front end" to the database server. When you query a database server, the query is processed on the server, and only the data you requested is returned to SQL Link.

Overview of SQL Link features

Following are some of the features SQL Link adds to Paradox. With Paradox 4.0 and SQL Link, you can

❏ run queries on SQL data using Paradox's powerful query-by-example (QBE) features

  - Paradox translates your query to a SQL statement you can display onscreen and save in a PAL script.
  - You can treat the resulting local table as you would any other Paradox table.
- create new tables on database servers and access existing remote tables
- manipulate data on database servers (add, copy, empty, and delete)
- create powerful PAL applications to access SQL data
- run SQL statements from within your PAL application or as a miniscript
- apply transaction processing techniques to protect the integrity of your SQL data
- handle errors that occur on database servers
- access multiple servers in a single Paradox session

What's changed in this version?

Since its last release, Paradox SQL Link has been improved in the following ways:

- When you connect to a remote database, Paradox no longer requires an additional user count for server connections.
- SQL Link now works with the run-time version of Paradox.
- SQL Link features an improved SQL Setup program (see Chapter 7 for details).
- The SQL Setup and UseSQL programs are now available from the ≡ (System) | Utilities menu.
- Paradox SQL Link supports new connections to DEC VAX Rdb/VMS, IBM's DB2, and Novell NetWare SQL.
- This version of Paradox SQL Link incorporates the latest vendor libraries.

How to use this book

The Paradox *SQL Link User's Guide* is a general guide for using SQL Link. You should also read the addendum describing your specific server environment, which guides you through installing SQL Link and troubleshooting problems.

Printing conventions

The Paradox manuals use special typefaces to help you distinguish between keys you press, names of Paradox objects, menu commands, and text you type. Table 1-1 lists these conventions.

Table 1-1 Printing conventions

Convention	Applies to	Examples
Bold	Any message displayed by Paradox	Paradox displays **Creating remote table...**
Italic	Tables, scripts, procedures, arrays, variables, glossary terms, emphasis, example elements	*Orders* table, *DoWait* procedure, the *Retval* variable, array *a*
ALL CAPS	DOS files and directories, reserved words, operators, PAL commands, SQL statements	PARADOX.EXE, LIKE, SORT
Initial Caps	Fields, menu commands, object names, applications, libraries	Stocks application, Price field, Tools | SQL
Keycap font	Keys on your computer's keyboard	*F2, Enter*
`Monospaced font`	PAL code examples	`SETKEY "F1" SORT TABLE() ON FIELD()`
Type-in font	Text that you type in	**=6/2/90, sampord**

In addition, Chapter 6 of this book uses the syntax notation described in Table 1-2.

Table 1-2 Syntax notation

Convention	Element	Examples	Meaning
ALL CAPS	Keyword	SHOWMENU	Type exactly as shown.
Italic	Fill-in	*TableName*	Replace with expression.
|	Choice	{ VARS | PROCS }	Choose *one* of the elements separated by the vertical bar.
[ ]	Optional	[OTHERWISE]	You *can* choose whether to include this.
{ }	Required	{ VARS | PROCS }	You *must* choose one of the elements separated by the vertical bar.

What to read before you use SQL Link

To install SQL Link on your network or workstation, read your server-specific addendum. Database administrators or Paradox users can install SQL Link.

In addition,

- To learn the basic concepts necessary to use Paradox with SQL Link, read Chapter 2.
- If you are new to Paradox, read Chapter 1 of the Paradox *Getting Started* manual and Chapters 1 and 2 of the Paradox *User's Guide* for a quick introduction.
- If you're familiar with Paradox, read Chapter 3 in this book for a brief hands-on tutorial of SQL Link's capabilities, and Chapter 5 for information on accessing remote data through the Paradox menus.
- If you are new to SQL, refer to the Glossary for an understanding of basic terminology.
- To create replicas for tables that already exist on your database server, read Chapter 7.
- If you are a database administrator, read this entire manual, paying particular attention to Chapters 4 and 7. Chapter 4 contains information on sending SQL statements directly to the server. Chapter 7 provides information on accessing tables that already exist on your database server and on customizing the communication links between Paradox and your database servers. In addition to this manual, you should also read your server-specific addendum.
- If you are a PAL application developer, you should read this entire manual to gain a full understanding of SQL Link, paying particular attention to Chapters 4, 6, and 8. In addition to this manual, you should also read your server-specific addendum.

How to contact Borland

Borland offers a variety of services to answer your questions about SQL Link. These services are available in every country where SQL Link is sold and vary from country to country. For the most up-to-date information on contacting Borland in your country, see the brochure on registering your product, which is included in your SQL Link package.

Registering your copy of SQL Link

To register your copy of SQL Link, complete and return your registration card. You can also register by calling 800-845-0147 (within the U.S. and Canada) or by contacting your local Borland representative (elsewhere).

When you register, you immediately become eligible for:

To register your copy
800-845-0147
(U.S. and Canada)

- Free access to Borland's world-class technical support. (Technical support is free unless the product was purchased under a program that specifies otherwise. You must pay telephone toll charges on technical support calls. Support, upgrade, and product information policies are subject to change. These terms and conditions are for U.S. and Canadian customers only.)
- Advance notice of new versions and special prices.
- Information on Borland database development and product enhancements, as well as information on other new Borland products.

Resources in your package

Your SQL Link package contains many resources to help you:

- The manuals provide information on every aspect of the program. Use them as your main information source.
- While using the program, you can press *F1* or click *F1* Help in the SpeedBar for help.
- Many common questions are answered in the README file located in the system files directory. Additional online information files on specific topics might also be included. The manuals refer to these files where appropriate.

Borland resources

Borland Technical Support publishes technical information sheets on a variety of topics and is available to answer your questions.

Techfax
800-822-4269 (voice)
(U.S. and Canada)

TechFax is a 24-hour automated service (available in the U.S. and Canada) that sends free technical information to your fax machine. You can use your touch-tone phone to request up to three documents per call.

File Download BBS 2400 Baud, 8, N, 1

The Borland File Download bulletin board system (BBS) has sample files, applications, and technical information you can receive by using your modem. No special setup is required.

Country	Modem number
U.S. and Canada	408-439-9096
Australia	(02) 953-9630

Online information services

Worldwide subscribers to the CompuServe, GEnie, or BIX information services can receive technical support by modem. Use the commands in the following table to contact Borland while accessing an information service.

Service	Command
CompuServe	`GO BORLAND`
BIX	`JOIN BORLAND`
GEnie	`BORLAND`

Address electronic messages to *Sysop* or *All*. Don't include your serial number; messages are in public view unless sent by a service's private mail system. Include as much information on the question as possible; the support staff will reply to the message within one working day.

Paradox Technical Support

Paradox Technical Support is available weekdays during business hours to answer any technical questions you have about Paradox.

Country	Phone number
U.S. and Canada	408-461-9155 6 a.m. to 5 p.m. Pacific time
U.K.	0734 320777
Australia	(02) 953-9500

Paradox Technical Advisor 900-555-1000 (voice) (U.S. only)

When you need an instant answer or a more advanced level of technical support (within the U.S.), you can call Borland's Technical Advisor service at 900-555-1000. You gain access within one minute. Each Technical Advisor call is $2.00 per minute (the first minute is free).

Information you need when you call support

When you call Technical Support, call from a telephone near your computer, and have the program running. Keep the following information handy to help process your call:

❐ product name, serial number, and version number

❐ the brand and model of any hardware in your system

❐ operating system and version number (Use the DOS command VER to find your DOS version number.)

- ❐ contents of your AUTOEXEC.BAT and CONFIG.SYS files (located in the root directory (\) of your computer's startup disk)
- ❐ a daytime phone number where you can be contacted
- ❐ if the call concerns a problem, the steps to reproduce it

You can also send support inquiries by fax or the postal system. For addresses and fax numbers, see the brochure on registering your product. Be sure to include the information listed in the preceding section.

Customer Service

Borland Customer Service is available weekdays during business hours to answer any non-technical questions you have about Borland products, including pricing information, upgrades, and order status.

Country	**Phone number**
U.S.	408-438-5300 7 a.m. to 5 p.m. Pacific time
Canada	416-229-6000
U.K.	0734 320022
Australia	(02) 953-9500

CHAPTER 2

Fundamentals

This chapter introduces you to the fundamentals of SQL Link. With an understanding of these basic terms and concepts, you'll be able to use Paradox more effectively to work with data on your database server.

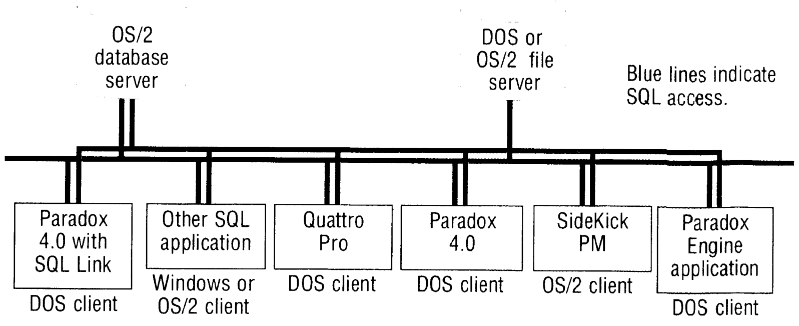
The SQL environment

SQL database servers are multiuser relational database management systems (RDBMS), based on what is called *client-server architecture*. In this environment, end-user programs like SQL Link are executed on *workstations*, while the RDBMS (including processing, integrity control, and security) is executed on the *database server*. Data is stored on a central machine (the database server), and users access the data from workstations (the *clients*) connected to that machine. Some database servers can run in a local area network (LAN) environment, such as OS/2, as well as in other multiuser environments, such as UNIX or VMS. If you would like to learn more about your database server and its specific environment, refer to your server manuals.

In an OS/2 LAN environment, a database server can double as a network file server or as a client, but is often a dedicated machine. As on a LAN without a database server, several people can access the same file at the same time to query the data, add or change information, and print reports. Only SQL applications can directly access SQL data. The SQL clients can, however, share the retrieved data with all other clients on the network.

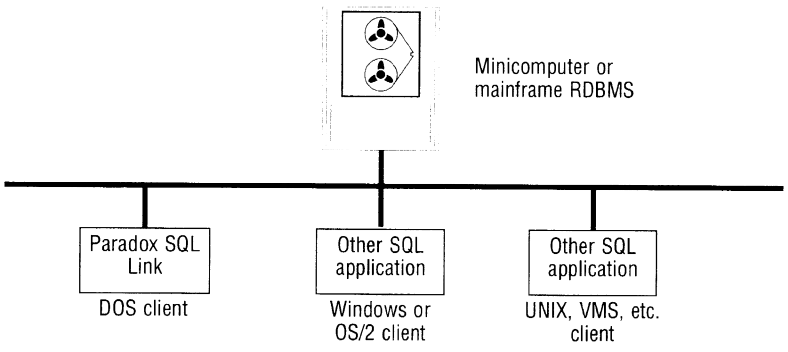
An example of a LAN server paired with a dedicated database server is shown in Figure 2-1.

Figure 2-1 Paradox SQL Link in a client/server system



Paradox SQL Link can also directly access a supported RDBMS on your minicomputer or mainframe, as shown in Figure 2-2. A file server (not shown) is also often included in this type of system.

Figure 2-2 Paradox SQL Link in a minicomputer or mainframe environment



Both SQL and Paradox databases are based on the *relational model.* Consequently, Paradox concepts and SQL concepts are very similar. On a SQL database server, information is organized in much the same way as in Paradox. Data is stored in *tables*. Tables consist of rows (*records*) and columns (*fields*). A table can have one or more indexes, which organize the information according to one or more fields (*key fields*) and make the data more quickly and easily accessible.

Note Throughout this manual, SQL tables are called *remote* tables, and Paradox tables are called *local* tables.

As in the Paradox multiuser environment, tables on database servers are often protected with passwords to prevent unauthorized access. SQL Link prompts you for your user name and password only when necessary. Once you're connected, you need the proper access rights—or, as they're known in most server environments, *privileges*—on your server to see or change data in a remote table. Your database administrator should give you details on your logon and access privileges.

The following sections describe some basic database server concepts that you might find helpful when working with SQL Link. You'll find server-specific information in the addendum accompanying this manual.

The database administrator

Managing a database server requires someone to fill the role of the *database administrator* (sometimes referred to as the *system administrator*). The database administrator is responsible for maintaining user accounts (user names, passwords, access privileges, and so on), managing storage, and ensuring the accuracy and integrity of data stored on the server. Because they are responsible for data integrity, they often provide guidelines for program development. Your database administrator can assist you with any questions you have about your server.

Transaction processing

SQL database servers handle requests in logical units or *transactions*. A transaction is a series of operations that must all be successfully performed before any changes to the database can be saved.

A good example of high-end transaction processing is an automated teller machine (ATM) transaction. Suppose you go to your ATM to transfer funds from your savings account to your checking account. Together these two operations are considered a single transaction because they're logically inseparable: You subtract a value from your savings account, then add that value to your checking account. You (and the bank) want to perform both of these operations or neither of them. If both operations succeed, then the transaction is successful; if either operation fails, the entire transaction fails.

Transaction processing ensures that your processing requests don't compromise data integrity. Database servers generally employ a write-ahead log to record changes you make to tables before actually writing them to disk. This log lets you accumulate related changes to remote tables until you're ready to save them. You can *commit*, or save, the changes when you're satisfied that no errors occurred during the transaction. If an error occurs, you can *roll back*, or abandon, the changes rather than save only some of the operations in the transaction. Issuing a COMMIT or ROLLBACK command concludes the current transaction.

SQL Link automatically commits changes immediately before and after certain operations (such as when you create, copy, or delete a table on the database server). Servers handle transaction processing in various ways. Some servers begin a new transaction automatically when you start your remote session or end a transaction, while others require that you do so explicitly. See your server-specific addendum to find out how your server handles transaction processing. You can also ask your database administrator to tell you when it's appropriate to use SQL Link commands to control transaction processing.

For more information about transaction processing commands, see the discussion on Tools | SQL | Transaction in Chapter 5, the "Transaction processing" section in Chapter 6, and your server manuals.

Nulls

SQL supports a field value referred to as *null*. Nulls have no explicitly assigned values; they are not blank values or zero values. Null values are not equal to any other value, including another null value. A null value means "value unknown."

You may come across a remote table that has been defined with null (and non-null) fields. A null field means that the field does not require a value. If an entry is not made, the value NULL is entered into the field automatically. A non-null field requires that a value be entered in that field. You'll get a server error message if you try to leave a non-null field blank when entering data into a remote table.

Note When SQL Link creates remote tables, it creates fields that allow nulls. If you don't want the remote table to allow nulls, create the table using passthrough SQL and replicate it using SQL Setup.

If you're a PAL programmer, see SQLVAL() in Chapter 6 for more information on how to handle nulls in PAL expressions.

Using Paradox to access SQL data

All programs in the SQL environment use SQL to communicate with database servers. Although different kinds of servers use different SQL dialects, SQL Link translates your requests into the appropriate SQL dialect. You can use SQL Link with different servers without learning all the SQL differences among server products. Special Paradox tables called *replicas* let you access remote tables in the same way you access local Paradox tables (see the discussion of replicas later in this chapter).

You don't have to learn SQL to access data on a SQL database server. Instead, you can use the Paradox menus and PAL to perform operations on remote tables.

Accessing existing data

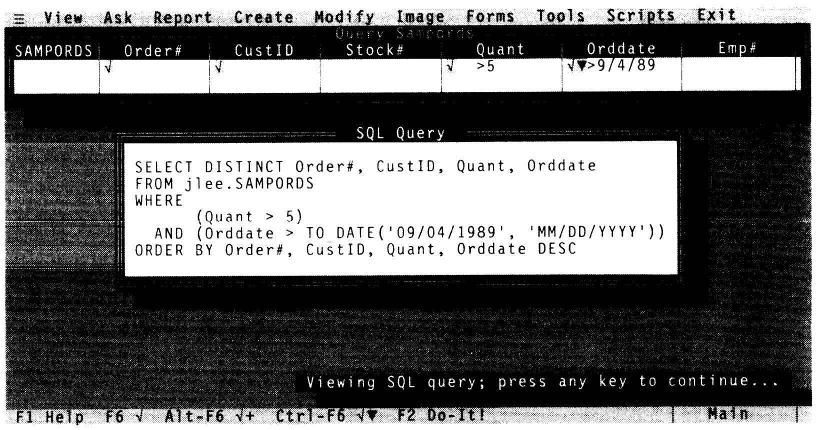
To access data that already exists on your server, use the SQL Setup Program to create replicas for the existing tables (see Chapter 7 of this manual). You can also access existing tables with passthrough SQL...ENDSQL commands or with UseSQL, the SQL command editor.

Querying remote tables

Querying a remote table is as easy as querying a local Paradox table: Choose Ask from the Paradox Main menu and select the remote table from a list. Next, fill out the Paradox *Query* form, specifying criteria to select the data you want to retrieve. Then, press *F2* Do_It! to process your query.

Any time you have a remote query on the workspace, *Alt-F2* ShowSQL shows you the SQL statement executed against the database server (see Figure 2-3).

Figure 2-3 The Show SQL Query screen


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Query Sampords

SAMPORDS	Order#	CustID	Stock#	Quant	Orddate	Emp#
	√	√		√ >5	√▼>9/4/89	

```
SQL Query

SELECT DISTINCT Order#, CustID, Quant, Orddate
FROM jlee.SAMPORDS
WHERE
      (Quant > 5)
  AND (Orddate > TO DATE('09/04/1989', 'MM/DD/YYYY'))
ORDER BY Order#, CustID, Quant, Orddate DESC
```

```
Viewing SQL query; press any key to continue...
```

F1 Help F6 √ Alt-F6 √+ Ctrl-F6 √▼ F2 Do-It!
Main

If you prefer to build queries in SQL, SQL Link lets you write SQL statements and send them directly to your server. For more information, see Chapter 4, the discussion on Tools | SQL | SQLSave in Chapter 5, and the discussion on SQL...ENDSQL in Chapter 6.

Replicas

You can still work with data stored in Paradox tables on your local hard disk, floppy disk, or file server (local tables). SQL Link also lets you access data in tables that reside on the database server (remote tables) in the same Paradox session.

To let you access remote tables as easily as local tables, SQL Link uses a special type of table called a *replica* to locate the remote table and to transfer data between Paradox and the database server. Replicas do not contain actual data. Instead, they contain the information necessary for SQL Link to connect to the database server on which the table resides, and to work with the data in that table.

In most cases, you can treat replicas as if they were regular Paradox tables. For example, to access a remote table from a Paradox menu, type its replica name or choose the replica from a list. If you choose it from a list, you'll see the replica name in the list box and the replica connection name highlighted below.

Similarly, to access a remote table from a PAL script, use its replica name in the PAL command. For example, to add records from a local table to a remote table, specify the replica name of the remote table:

```
ADD "Loctable" "Myreplca"
```

Creating remote tables

Whenever you use SQL Link to create a remote table on your database server (either by choosing Create or Tools | Copy from Paradox menus, or by using CREATE or COPY in a PAL script), SQL Link automatically creates a replica of it with the same name.

Table names on database servers can be more than eight characters. Due to DOS limitations, replica names cannot exceed eight characters. When you use SQL Setup to create replicas for existing SQL tables, SQL Link truncates remote table names to eight characters and uses this name for the replica. If more than one table on the server shares the same name or the same first eight characters, SQL Link resolves the problem by truncating the duplicate names, then adding a hyphen and a sequential number. SQL Link provides you with a list of tables it renames in this manner.

For more information about replicas and accessing tables that already exist on your server, see Chapter 7.

Connecting to the server

SQL Link manages the communication session with your server so you can focus your attention on working with your data. SQL Link connects to the server automatically through replicas whenever you need to access remote data. You don't even need to know where the table is located; SQL Link makes the server connection and finds the table for you. This type of connection is called a *replica connection.*

Before you create a new remote table (for example, when you use Tools | Copy to create one) or use SQL Setup to create replicas for existing SQL tables, however, you need to explicitly connect to the server so that SQL Link can communicate with the server and create the necessary replica. Except for these operations, you don't need to explicitly select a server connection, as the replica connection can be used.

Default server connection

SQL Link comes with a default server connection for each supported server, and displays its title and description when you work with a remote table. A *connection* identifies the type of server and its SQL dialect.

Additionally, each server connection contains information that you supply, such as user name and server name. Your server requires these *connection parameters* for access. SQL Link prompts you for your user name and password only when

- ❐ You connect to a server for the first time in a session.
- ❐ You change connections to access data in a different location.

For more about the connection parameters for your particular server, see your server-specific addendum. Your database administrator can assist you with the exact information required to access your database server.

Although you can use the standard connections that come with SQL Link, you can also create custom connections for individual users, groups of users, or different applications. For example, you might want to create a connection to let the Marketing Department access the Sales Department data. To add a custom connection to the list of server connections, use the SQL Setup Program. For more information, see Chapter 7.

Access privileges

You must have the proper *access privileges* on your server to use remote tables. Database servers support sophisticated security with different levels of access based on your user name, group, and password. Neither SQL Link nor Paradox can override your server's security.

Table 2-1 lists the specific privileges you need on your server for each remote Paradox operation.

Table 2-1 Server privileges needed for remote operations

Paradox command	Server privileges needed
Ask	SELECT
INSERT query	INSERT
DELETE query	DELETE
CHANGETO query	UPDATE

Create|Remote	CREATE TABLE
	CREATE INDEX *

Modify|DataEntry	INSERT
Tools|Delete|Table	DROP TABLE DROP INDEX*

Tools|More|Add	SELECT
	UPDATE
	INSERT

Paradox command	Server privileges needed
Tools|More|Empty	DELETE
Run SQLSetup	SELECT

* Needed only if you are creating indexed tables

SQL Link requires additional privileges on some servers. See your server-specific addendum for information.

To maintain security on your server, SQL Link prompts you for your user name and password when necessary. If you don't have access privileges for a particular operation, the database server won't let you proceed, and SQL Link provides a message to inform you of the problem. To obtain the necessary access privileges on your database server, see your database administrator.

Paradox/server differences

Your database server might have different rules for naming tables and fields, and might support different field types than Paradox. For example, even though you can use a hyphen (–) in a Paradox table name, some servers don't allow this character in a remote table name. Similarly, you can use spaces in a Paradox field name, but not in a remote table field name. In addition, some servers may be case sensitive to table and field names. Paradox always creates tables using all uppercase letters, and field names using initial uppercase letters. You need to conform to the table- and field-naming rules of your database server *as well as* Paradox rules when you create a new remote table using SQL Link. The most common differences are covered in detail in your server-specific addendum.

Note Memo field types in remote tables are not supported; remote fields longer than 255 characters are truncated to 255. Other restrictions apply to certain servers; see your server-specific addendum for details.

Short number and currency field types

Not all servers support the Paradox short number (S) or currency ($) field types. If you create a replica in SQL Setup for a remote table on a server that does not support currency field types, for example, SQL Link converts Paradox field types to the most similar supported field type. It then prompts you to decide if it should interpret that table's numeric columns as containing currency values.

If you are creating a remote table, SQL Link creates a replica that allows it to map local to remote field types. Even though Paradox can treat remote numeric data as currency values, the data in the remote table is never affected by this mapping.

Rules for indexing

Your database server might also have different rules for creating indexes than those used by Paradox. A Paradox-compliant index is a unique index based on one or more consecutive fields (*keyfields*), beginning with the first field in the table and proceeding consecutively to the last field in the index. For local tables, Paradox uses one main index (the *primary key*). In contrast, on most database servers you can specify any field or group of fields as an index (unique or non-unique), regardless of its location or whether the fields in the index are consecutive.

Indexes are used to identify records and ensure that records added to or updated in a table won't have the same value in key (or uniquely indexed) fields. See the Paradox *User's Guide* for more on how Paradox uses indexes to update Paradox tables.

If unique index violations occur (which Paradox calls *key violations*) when you add records to a remote table with a unique index, SQL Link produces a local *Keyviol* table containing the duplicate records. (Rows which the server does not allow you to insert or update in a server table for other reasons (such as values out of range, and so on) are placed in a *Problems* table.

Note Some database servers do not return a specific error for insert or update failures due to unique index violations. In that case, the rows that could not be inserted or updated are placed in a *Problems* table.

When you create an index for a server table at the same time you create the remote table (using Create | Remote), you can only create indexes that follow Paradox rules for defining indexes on consecutive key fields of the table. You can still create other types of indexes on remote tables using the SQL...ENDSQL command to directly execute SQL statements (such as CREATE INDEX) against a particular server. For example, you might want to create other types of indexes to improve the performance of queries. Only those indexes that comply with Paradox indexing rules become part of a replica's structure if you use SQL Setup to create a new replica on the server table.

When using SQL Setup, SQL Link evaluates the unique indexes available on the server for a particular table (if any exist). The criteria that SQL Link uses in selecting a unique index to add to a server table replica is the following:

1. Unique indexes used by Paradox must be based on consecutive fields, starting with the first field of the server table. The index must include all fields from the first field in the table structure to the last field included in the index. (Indexes on memo fields are not allowed.)
2. If more than one unique index qualifies, the index based on the fewest number of fields is selected.

Having a unique index available on a remote table is only required for table updates (using Tools | More | Add | Update), since Paradox uses the index to uniquely identify a row. You can perform all other operations, such as creating and displaying queries, and adding or deleting rows, without an index.

Working with remote tables

A database server processes your requests in a different manner than Paradox. At times you won't have the same level of control over remote operations that you have in the normal Paradox network environment. The following sections describe some of these situations.

Performance

While Paradox delivers your request to the server, the server decides the order in which it processes your request. Several factors affect the speed with which a database server returns results:

- the number of users accessing data on the server at the time you make your request
- the level of traffic on the network and on the communication link between Paradox and the server
- the degree of multiuser concurrency supported by your server
- the size of the tables you are accessing and the complexity of your request
- use of indexed fields in queries, joins, and so forth
- user priority (queues)

If you think your remote operation is taking too long, press *Ctrl-Break* to stop it.

Queries

When you're querying remote tables, you rely on the database server to process your request along with the requests it receives from all other users. Because the database server can give you access to vast amounts of data, you should be aware of the limitations of your local environment. You might want to do some aggregate queries first to estimate the size of a remote table. For example, a CALC COUNT ALL query returns the number of records in the remote table, and a CALC COUNT returns the number of records that your selection would report.

Normally, SQL Link generates your SQL statements for you, but you can create your own and send it directly to the server by enclosing it in a SQL...ENDSQL command (a technique called *passthrough*).

There are several reasons why the database server might not process your query at all:

- The server does not support your query statement. In this case, you'll get an immediate message from SQL Link.

- The structure of the remote table has changed since you created the replica, and you'll need to regenerate the replica (see Chapter 7 of this manual for more information). In this case, you'll get a server error message.
- You don't have sufficient access privileges. In this case, you'll get a message from the server. Consult your database administrator about obtaining the appropriate privileges.

Changed tables

When you perform a CHANGETO query in QBE, a *Changed* table is not created as it would be in native Paradox.

Deleted tables

When you perform a DELETE query in QBE, a *Deleted* table is not created as it would be in native Paradox. To find out how many records will be deleted by a DELETE operation, run the QBE DELETE query against the remote table, but omit the DELETE keyword in the leftmost column.

Inserted tables

When you perform an INSERT query in QBE, an *Inserted* table is not created as it would be in native Paradox.

"FAST" queries

Paradox 4.0 has a FAST keyword, which skips making *Changed, Inserted,* or *Deleted* tables for those types of queries. Since these tables are never created for remote queries, this keyword has no effect.

Concurrency

Your server controls all *implicit locks*. Therefore, your database server might deny you access to remote data because another user is accessing the same remote tables you want to access. In these instances, SQL Link must wait until the user or the server releases the locks on those tables. *Deadlock* occurs when two users hold locks on data required by the other; neither transaction can complete before the other's tables or records are unlocked. Each server has its own deadlock detection and recovery procedure that, in most cases, results in one of the user's queries being canceled. For more information on deadlock recovery procedures, refer to your server manuals.

Errors on the database server

To help you find the source of errors that occur during a server transaction, SQL Link returns an error message and an error code (error codes are returned only to PAL applications). For errors specific to your database server, see your server manuals to find out how to correctly process your query.

If the remote error occurs while you're running a PAL script, you can recover from the error from within your application using the normal PAL error-handling methods. SQL Link lets you retrieve the error codes and messages from the database server (as well as Paradox's own error messages).

For more information on how to handle these messages, see the "Error handling" section in Chapter 6 of this manual.

Using PAL to access SQL data

You can enhance existing PAL applications to take advantage of SQL. For example, you can share data with other users and applications; you can cut down on network traffic by sending only SQL requests and *Answer* tables over the network; and you can enable users to connect to multiple servers during a single Paradox session. For more information, see Chapter 6.

Finding out more about your server

Throughout this manual, you'll be directed to your database administrator or server manuals to learn how your database server handles certain operations. Database servers function differently in particular circumstances. Some of the more important differences are discussed in your server-specific addendum.

Before you begin using SQL Link, you should obtain the following information about your server:

- a list of the query operations that your server supports
- if your server begins a transaction automatically, or requires that you do so explicitly
- a list of error codes and messages that your server produces when an error occurs
- your access privileges on the server, including your user name and password

You can find most of this information in your server-specific addendum. In addition, you should learn how your server resolves deadlocks by referring to your server manuals.

CHAPTER 3

A quick tour

This chapter takes you on a quick, hands-on tour of the Paradox SQL Link menus. To take this tour, you should be familiar with the Paradox menus and should have already read Chapter 2 of this manual. If you are new to Paradox, read Paradox *Getting Started* and Chapters 1 and 2 of the Paradox *User's Guide*. You should spend some time using Paradox before working with SQL Link.

This tour assumes that SQL Link is installed and running, and that it can connect to the database server from your workstation. If you need to install SQL Link, see the installation instructions in your server-specific addendum. To test the connection, use your server's DOS-based diagnostic tools. Finally, ensure that you have privileges to create, update, and delete files on your server. See your database administrator to obtain the appropriate access privileges.

In this tour, you will use the Paradox menus to

- ❒ connect to your database server
- ❒ create a remote table on your database server
- ❒ add data to the remote table
- ❒ query the remote table
- ❒ create a report of data on the remote table
- ❒ copy the remote table to a local table
- ❒ empty the remote table
- ❒ delete the remote table

This quick tour covers only a few of SQL Link's features. For a detailed discussion of each menu command, see Chapter 5 of this manual. If you want to learn about using SQL-specific PAL commands and functions, see Chapter 6. To learn how to use the SQL Setup Program to access a remote table that was created with a program other than Paradox, see Chapter 7.

Starting Paradox SQL Link

Before starting Paradox, make sure you load the network communication program required by your server (see your server-specific addendum for details), and make sure your server is running.

With SQL To start SQL Link, start Paradox from the Paradox system files directory (usually PDOX40) with the following command:

paradox

With international date format If your server is an international (non-U.S.) server, specify the international switch when you start Paradox

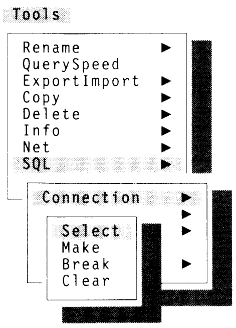
paradox -intlsql

This switch configures Paradox to convert dates to the international date format DD.MM.YYYY. For more information, see "International date format" in your server-specific addendum.

Without SQL When you start Paradox, SQL Link is automatically activated. If you want to run Paradox without using its SQL capabilities, however, start Paradox with the following command:

paradox -sql off

Selecting a server connection

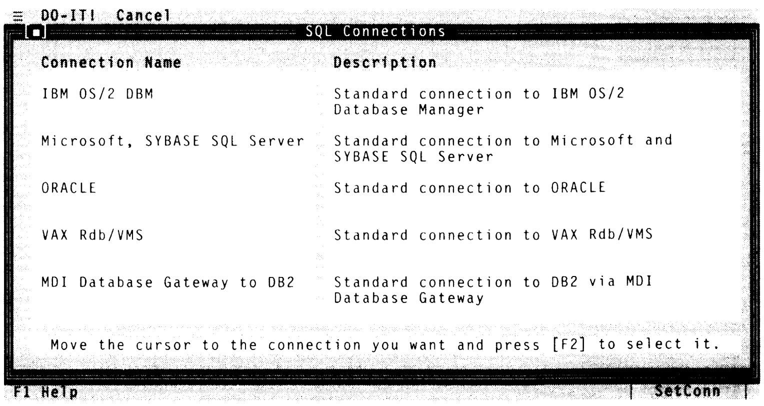


Before you create a remote table, you'll need to connect to the server on which the table will reside. You need to set an explicit server connection only when you're creating a new remote table.

To select a server connection,

1. Choose Tools | SQL | Connection | Select from the Main menu.

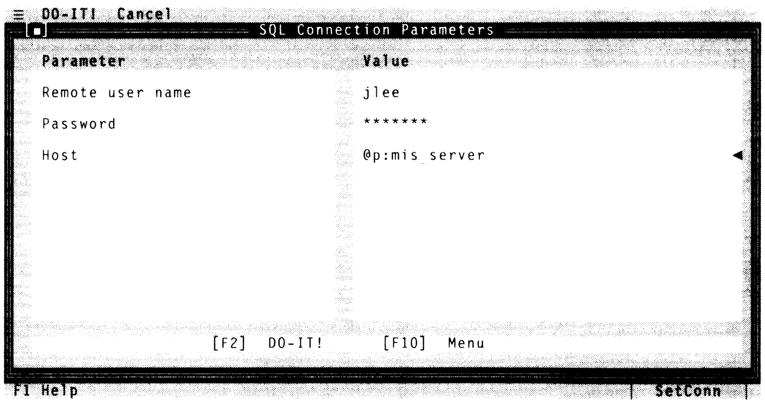
   SQL Link displays a list of available server connections. Only those products that you installed (or customized with SQL Setup) are displayed in this list.


```
≡ DO-IT!  Cancel
                              SQL Connections
 Connection Name                  Description

 IBM OS/2 DBM                     Standard connection to IBM OS/2
                                  Database Manager

 Microsoft, SYBASE SQL Server     Standard connection to Microsoft and
                                  SYBASE SQL Server

 ORACLE                           Standard connection to ORACLE

 VAX Rdb/VMS                      Standard connection to VAX Rdb/VMS

 MDI Database Gateway to DB2      Standard connection to DB2 via MDI
                                  Database Gateway

  Move the cursor to the connection you want and press [F2] to select it.
F1 Help                                                          | SetConn |
```

2. Move the cursor to the server connection you want, then press *F2* Do_It! to select the connection.

This screen differs depending on the database server you're connecting to. See your server-specific addendum for information on connection parameters.

3. If SQL Link requires additional information for the connection (such as your user name, password, or server name), type it in, and press *F2* Do_It! when you're finished.


```
≡ DO-IT!  Cancel
                        SQL Connection Parameters
 Parameter                         Value
 Remote user name                  jlee
 Password                          *******
 Host                              @p:mis_server                          ◄




                      [F2]  DO-IT!       [F10]  Menu
F1 Help                                                          | SetConn |
```

For more about selecting a server connection, see the discussion of Tools | SQL | Connection | Select in Chapter 5. For more information on the connection parameters required by your server, see your server-specific addendum and your server manuals.

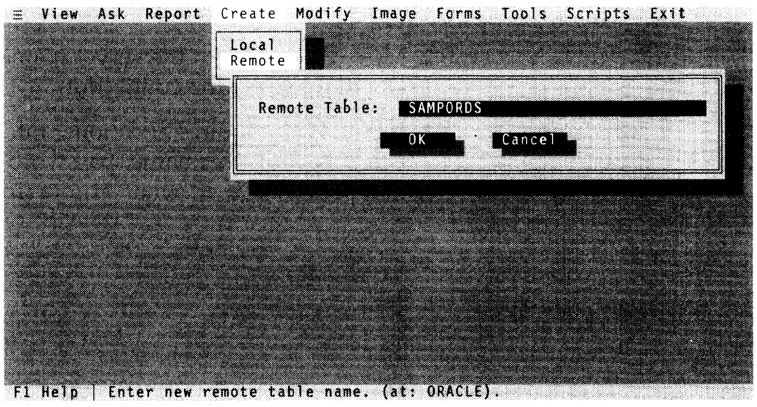
Creating a remote table

```
Create
 Local
 Remote
```

Now you're ready to create a new table on the database server. Creating a remote table is the same as creating a local Paradox table, except the rules for field names and field types in remote tables differ from those in Paradox tables. For example, servers don't permit spaces in field names. For more information on the rules for your server, see your server-specific addendum and your server manuals. You *cannot* use Paradox or server-specific reserved words for table or field names. See Appendix C of the *PAL Reference* and your server manuals for a complete list of reserved words.

To create a remote table, follow these steps:

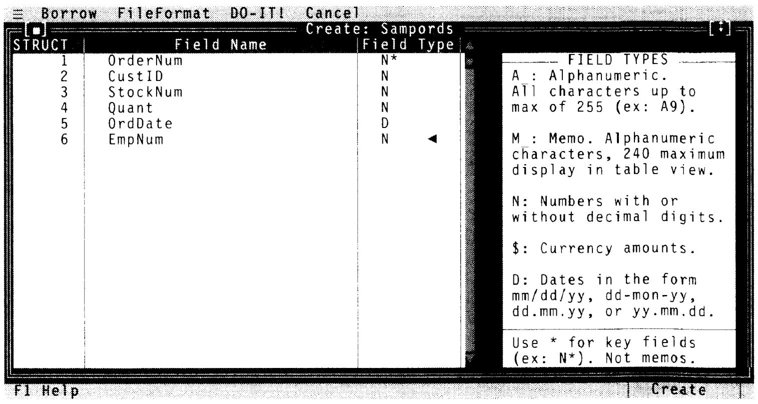
1. Choose Create from the Main menu.
2. There are two options: Local and Remote. Choose Remote to create a remote table.
3. For the new table name, type **sampords** and press *Enter.*


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Local
Remote
OK
Cancel

```
Remote Table: SAMPORDS
```

```
F1 Help | Enter new remote table name. (at: ORACLE).
```

4. Fill in the table structure image as shown in the next figure:



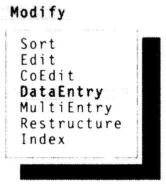
```
STRUCT | Field Name   | Field Type
   1   | OrderNum     | N*
   2   | CustID       | N
   3   | StockNum     | N
   4   | Quant        | N
   5   | OrdDate      | D
   6   | EmpNum       | N
```

```
———— FIELD TYPES ————
A : Alphanumeric.
All characters up to
max of 255 (ex: A9).

M : Memo. Alphanumeric
characters, 240 maximum
display in table view.

N: Numbers with or
without decimal digits.

$: Currency amounts.

D: Dates in the form
mm/dd/yy, dd-mon-yy,
dd.mm.yy, or yy.mm.dd.

Use * for key fields
(ex: N*). Not memos.
```

When you're finished, press *F2* Do_It! to create this remote table and its index. SQL Link also creates its local replica (containing structural and connection information) so it can find the remote table later.

For more information about creating a remote table, see the discussion of Create in Chapter 5 of this manual. For general information about creating tables, see Chapter 10 of the Paradox *User's Guide*.

Entering data into a remote table

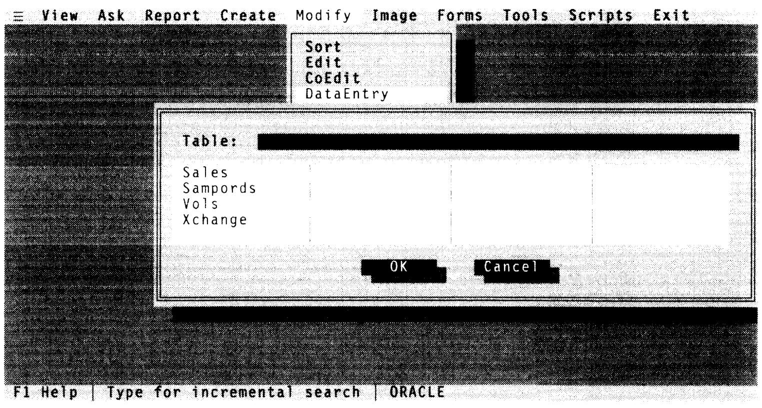


Choose Modify | DataEntry to enter data into a remote table. You can enter data into a remote table just as you enter data into a local Paradox table. SQL Link even detects key violations and puts duplicate records in a local *Keyviol* table. If any other problems occur (such as attempting to add a null value to a field in a remote table that doesn't allow nulls), SQL Link generates a local *Problems* table.

Note When Paradox creates remote tables, it creates columns that allow nulls. If you don't want the remote table to allow nulls, create the table using passthrough SQL and replicate it using SQL Setup.

Now enter data into the remote table you've just created:

1. Choose Modify | DataEntry from the Main menu.
2. Press *Enter* to display a list of tables.


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Sort
Edit
CoEdit
DataEntry
Table:
OK
Cancel

Sales
Sampords
Vols
Xchange

F1 Help | Type for incremental search | ORACLE

   Select *Sampords*. When you select the remote table, SQL Link displays the server connection it uses to find the remote table in the message line at the bottom of the screen. Press *Enter* to choose *Sampords*.

3. When you see the blank *Entry* table that corresponds to *Sampords*, enter the records with the information shown in the following table.

Table 3-1 Sampords

OrderNum	CustID	StockNum	Quant	Orddate	EmpNum
2280	4277	130	1	4/22/87	775
3351	3266	519	1	12/16/90	422
8070	6125	632	8	6/04/90	146
6235	2779	890	1	8/01/88	517

4. When you're finished, press *F2* Do_It! to add these records to the remote table.

For more information about entering data into a remote table, see the discussion of Modify | DataEntry in Chapter 5.

Querying a remote table

≡ View Ask Report Create Modify Image Forms Tools Scripts Exit

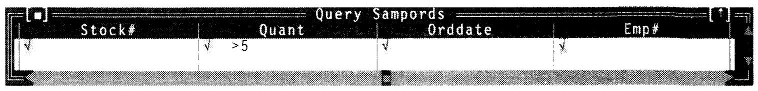
You can now query the data you've entered in this remote table. If the database server finds records that satisfy your query, SQL Link produces a local *Answer* table, which you can treat like any other *Answer* table.

SQL Link lets you perform sophisticated Paradox queries on remote data. Not all database servers support the full range of query operators available in SQL Link. For more information, see your server manuals and Tables 5-1 and 5-2 in Chapter 5 of this manual.

Query the data you've just entered by following these steps:

1. Choose Ask from the Main menu.
2. When SQL Link asks you to enter a table name, type **sampords** or choose it from the list box.
3. Fill in the *Query* form as shown in the following figure. (To select all records and all fields in *Sampords*, press *F6* Checkmark with the cursor in the leftmost field.)

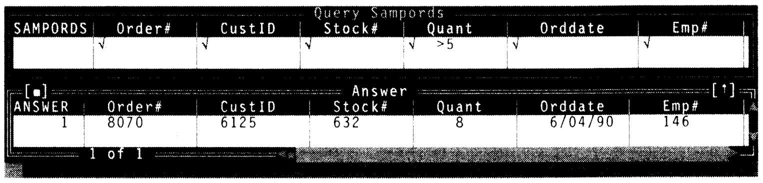
This query selects records with Quant > 5.



Query Sampords

Stock#	Quant	Orddate	Emp#
√	√ >5	√	√

4. When you're finished, press *F2* Do_It! to run this query. You now see only those orders with quantities greater than five.



```
                              Query Sampords
SAMPORDS | Order#   | CustID   | Stock#   | Quant    | Orddate   | Emp#
         | √        | √        | √        | √ >5     | √         | √

[■]                            Answer                                [↑]
ANSWER   | Order#   | CustID   | Stock#   | Quant    | Orddate   | Emp#
       1 | 8070     | 6125     | 632      |        8 |   6/04/90 | 146
         1 of 1
```

When you're finished viewing the *Answer* table, press *Ctrl-F8* WinClose to remove it from the workspace.

For more information about querying a remote table, see the discussion of Ask in Chapter 5 of this manual. For general information about querying tables, see Chapter 5 of the Paradox *User's Guide*.

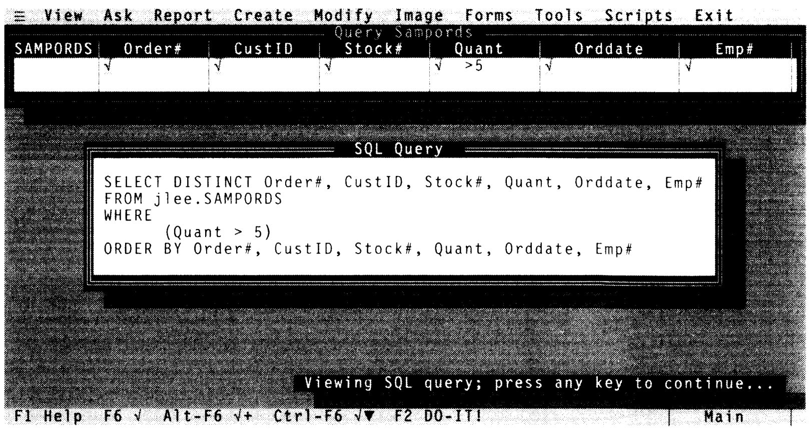
Viewing the SQL translation of your query

Alt-F2
ShowSQL

SQL Link translates your remote query into the appropriate SQL dialect for your server. Once you've written your query, you can press *Alt-F2* ShowSQL if you want to see the SQL statement the server receives and processes.

To see the SQL translation of the query you just ran in the previous example, press *Alt-F2* ShowSQL.

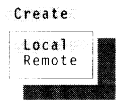
SQL Link displays your query as shown in this figure:



```
≡ View  Ask  Report  Create  Modify  Image  Forms  Tools  Scripts  Exit
                              Query Sampords
SAMPORDS | Order#   | CustID   | Stock#   | Quant    | Orddate   | Emp#
         | √        | √        | √        | √ >5     | √         | √

                    SQL Query
  SELECT DISTINCT Order#, CustID, Stock#, Quant, Orddate, Emp#
  FROM jlee.SAMPORDS
  WHERE
        (Quant > 5)
  ORDER BY Order#, CustID, Stock#, Quant, Orddate, Emp#

                          Viewing SQL query; press any key to continue...
F1 Help  F6 √  Alt-F6 √+  Ctrl-F6 √▼  F2 DO-IT!                  | Main
```

Once you're finished viewing the query, press any key to clear it from the workspace.

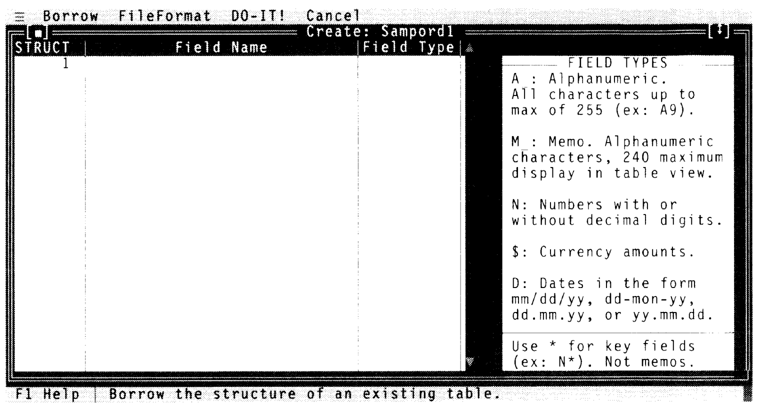
Adding records from one table to another



You can add all the records from a local table to a remote table or vice versa, as long as the fields in the two tables are compatible. If the target table is indexed and key violations occur, SQL Link produces a *Keyviol* table.

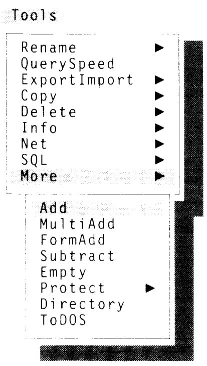
Create a local table (to be the target table) named *Sampord1* with a structure identical to *Sampords* by borrowing *Sampords'* structure:

1. Choose Create | Local from the Main menu to create a local table.
2. For the new table name, type **sampord1** and press *Enter*.
3. Press *F10* to display the Create menu.



```
≡ Borrow  FileFormat  DO-IT!  Cancel
[■]                    Create: Sampord1                         [↕]
STRUCT |        Field Name          |Field Type|
     1                                              FIELD TYPES
                                              A : Alphanumeric.
                                              All characters up to
                                              max of 255 (ex: A9).

                                              M : Memo. Alphanumeric
                                              characters, 240 maximum
                                              display in table view.

                                              N: Numbers with or
                                              without decimal digits.

                                              $: Currency amounts.

                                              D: Dates in the form
                                              mm/dd/yy, dd-mon-yy,
                                              dd.mm.yy, or yy.mm.dd.

                                              Use * for key fields
                                              (ex: N*). Not memos.
F1 Help | Borrow the structure of an existing table.
```

4. Choose Borrow from the Create menu.
5. Type **sampords** or choose it from the list box.
6. Press *F2* Do_It! to create the local table.



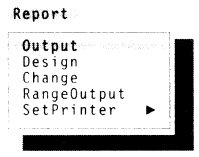
Next, copy all the records from the remote table *Sampords* to the local table *Sampord1*.

1. Choose Tools | More | Add from the Main menu.
   - ❒ For the source table, type **sampords**, or choose it from the list box.
   - ❒ For the target table, type **sampord1**, or choose it from the list box.
2. Choose NewEntries to add the records.

SQL Link adds all the records from the remote table to the local table and displays the local table *Sampord1*.

Suppose you want to add the records from *Sampord1* back to *Sampords*. If you choose NewEntries, SQL Link produces a *Keyviol* table because there are duplicate key values. If you choose Update, SQL Link replaces the records in *Sampords* with all the records from *Sampord1*. For more information about adding records to another table, see the discussion of Tools | More | Add in Chapter 5 of this manual.

Reporting on data in a remote table



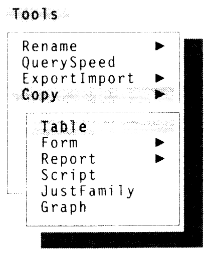
You can use Paradox and SQL Link to design and print reports on data in a remote table.

To print a standard report for the remote table *Sampords*,

1. Choose Report | Output from the Main menu.
2. For the table name, type **sampords**, or choose it from the list box.
3. Choose R to print the standard report.
4. Choose Printer to send the report to the printer or Screen to view the report onscreen.

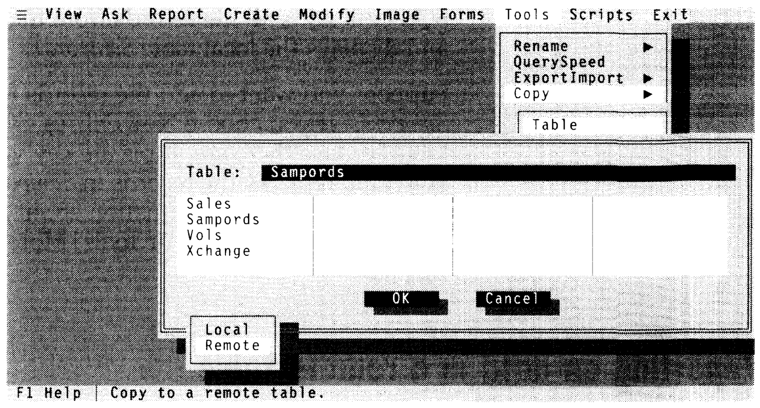
When you print a report to the screen, you can select Cancel | Yes or click on the close box to stop viewing the report. For more about reporting on remote tables, see the discussion of Report in Chapter 5 of this manual.

Copying a remote table



You can copy data to or from a remote table. Follow these steps to copy the structure and data of the remote table *Sampords* to a new remote table named *Sampord2*:

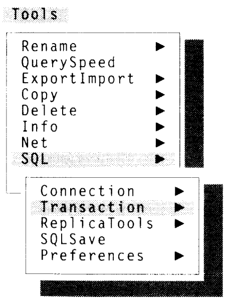
1. Choose Tools | Copy | Table from the Main menu.
2. For the source table name, type **sampords** or choose it from the list box.
3. Choose Remote to copy to a remote table.



4. For the target table name, type **sampord2** and press *Enter*.

If you copy a local table to a remote table, the local table's naming conventions must conform to your server's rules (see your server-specific addendum). For more information about copying tables, see the discussion of Tools | Copy in Chapter 5 of this manual.

Transaction processing



SQL Link makes use of the transaction-processing facilities of your database server, allowing you to define a transaction. For each transaction, you determine whether you want to commit the changes (if the operations were successful), or roll back the changes (if errors occurred during processing). Servers handle transaction processing differently; some start a transaction for you automatically while others do not. See your server-specific addendum for more information.

SQL Link normally commits each change to a remote table automatically. To do this manually, you need to turn AutoCommit off. Here's how:

1. Choose Tools | SQL | Preferences | AutoCommit from the Main menu.
2. Choose No to turn AutoCommit off.

Next, choose Tools | SQL | Transaction | Start to start a transaction. On servers that don't require that you start transactions explicitly, this command is ignored. Now, add two new records to the remote table *Sampords*:

1. Choose Modify | DataEntry from the Main menu.

2. Type **sampords** for the name of the table.
3. Enter two new records.
4. Press *F2* Do_It! to add the records.

At this point, you haven't committed the changes. If you query the table, however, you see the new records you've added. To save these new records in the remote table, choose Tools | SQL | Transaction | Commit from the Main menu. SQL Link saves your changes on the server and ends the transaction.

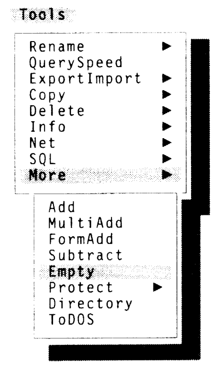
Now suppose you decide to abandon the changes rather than save them. Start a new transaction and add two more records to the table. When you query the table, you see the new records you've added. To roll back these changes, choose Tools | SQL | Transaction | RollBack from the Main menu. Query the table again. The new records will not appear in the *Answer* table.

Important Remember to reset AutoCommit to Yes after this exercise.

1. Choose Tools | SQL | Preferences | AutoCommit from the Main menu.
2. Choose Yes to turn AutoCommit on.

For more information, see "Transaction processing" in Chapter 6 of this manual.

Emptying a remote table



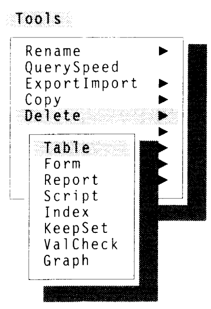
You can remove all the records from a remote table. You would do this if you want to keep the structure of a remote table but no longer need the data it contains.

To empty a remote table,

1. Choose Tools | More | Empty from the Main menu.
2. For the table name, type **sampords** or choose it from the list box.
3. Choose OK to confirm and delete all records.

For more information about emptying tables, see the discussion of Tools | More | Empty in Chapter 5 of this manual.

Deleting a remote table



You can delete a remote table you no longer need on your database server. When you delete a table, whether local or remote, you also delete any associated files, such as its indexes and forms. If it's a remote table, you delete its replica as well.

To delete a remote table,

1. Choose Tools | Delete | Table from the Main menu.
2. For the table name, type **sampords** or choose it from the list box.
3. Choose OK to confirm and delete the table.

Repeat these steps for *Sampord2.*

For more information about deleting remote tables, see the discussion of Tools | Delete in Chapter 5 of this manual.

CHAPTER 4

UseSQL, the SQL command editor

This chapter introduces you to the tool you can use to send SQL statements directly to your server. UseSQL is the command editor you use to write and execute SQL statements in SQL Link.

You can play the *UseSQL* script anywhere in Paradox or select ≡ | Utilities | UseSQL from the Main menu. Although the Paradox SQL Link menus and PAL commands provide most of the functionality you need to work with remote data, some tasks, such as system administration, must be performed using SQL directly. That's where UseSQL comes in. With UseSQL, you can

- ❒ create and test SQL statements before including them in a PAL program
- ❒ save a set of frequently used SQL statements and run them by picking individual statements from a list

UseSQL cards

You can think of UseSQL as providing a sequence, or stack, of cards containing SQL statements. The length of each statement is unlimited, but there can be only one statement per card. Each card is numbered, and you can see only one of them at a time. You execute each statement individually by displaying it onscreen and pressing *F2* Do_It! or selecting DO-IT! from the menu. The cards are arranged so that if you're at the last card in the stack and ask to see the next card, the first card appears.

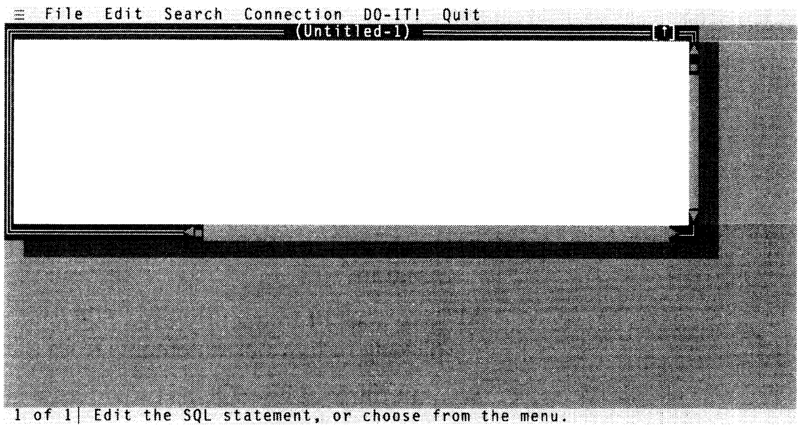
UseSQL provides a command editor so that you can cut and paste single lines or entire SQL statements from one card and paste them to another. You can try variations of the same statement by copying it to different cards and changing certain values. The SQL command editor functions identically to the Paradox Editor. It provides all the tools you need to write single SQL statements, or to write and debug full SQL programs.

SQL statements can span multiple lines, and the number of cards you can have is limited only by available disk space. Each UseSQL card can contain only a single SQL statement and only one card can be active at any time.

Starting UseSQL

UseSQL is a PAL script that you play from the ≡ | Utilities menu by choosing ≡ | Utilities | UseSQL. You can also choose Scripts | Play and select the *UseSQL* Script.

UseSQL displays the last card you used in the previous session from the card stack in your working directory. The first time you start UseSQL, there are no cards in the stack, so UseSQL displays an empty card. You can now type in any valid SQL statement, using the appropriate dialect for your database server.



```
1 of 1| Edit the SQL statement, or choose from the menu.
```

Connection menu

The Connection menu has the following options:

Table 4-1 UseSQL Connection menu options

Menu option	Description
Select	Lets you specify a server connection (if you have not already chosen one using Tools|SQL|Connection|Select from the Main menu). If you have established a connection but want to change it, you can use this option to break the current connection and establish a new one.
Make	Re-establishes the server connection stored in memory after you perform Connection|Break or break the connection some other way (for example, by pressing *Ctrl-Break* during a query).

Menu option	Description
Break	Allows you to break the current server connection without immediately re-establishing another connection. Information about the connection remains in memory.
Status	Displays one of three different messages indicating the status of a connection: 1. **Connected to** ***<servername>*** (if you're connected to a server) 2. **Connection set to** ***<servername>*** (if you've specified a server connection but aren't currently connected) 3. **Not connected to a server** (if you haven't specified a connection)

File menu

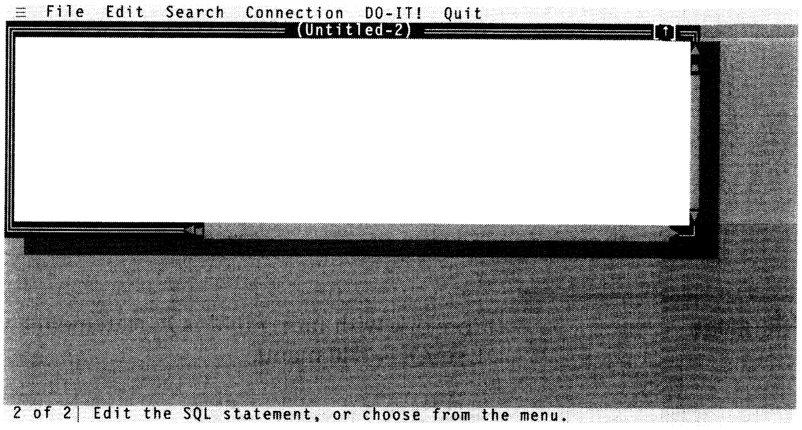
To work with individual SQL statements, you access options in UseSQL's File menu.

Table 4-2 File menu options

Menu option	Description
New	Lets you create a new SQL statement.
Open	Lets you access a different SQL statement. When you choose this option, the Select A Card dialog box appears, listing all of the SQL statements you've entered. It also lists the user name of the SQL statement's creator, the date on which the statement was created, the date on which the statement was last run, and the connection for which it was created. You can move to another card by selecting it from the list.
Save	Lets you save the current SQL statement and continue editing.
CopyTo	This option is context sensitive, depending on whether a card or an *Answer* table is in the active window. If the active window contains a card, CopyTo lets you save the current SQL statement to a script file (with the statement nested in a SQL...ENDSQL command and with a .SC extension) or to a text file (containing only the statement). If the active window contains an *Answer* table, CopyTo lets you save a copy of the table under a different name.
InsertFile	Inserts a file at the location of the cursor.
WriteBlock	Prompts you for a file name then writes the text you have selected on the current UseSQL card to a file.
Print	Lets you print the current SQL statement to your printer.

Creating a SQL statement

To create a new SQL statement, choose File | New. When you choose OK, a new card appears. You can now type in any valid SQL statement, using the appropriate dialect for your database server.



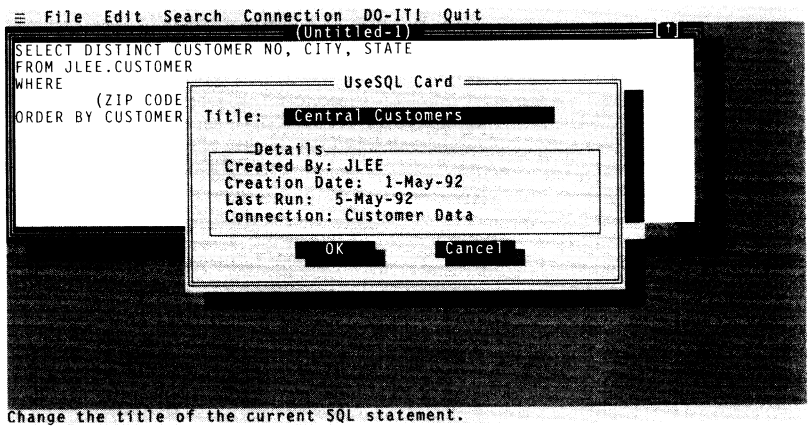
2 of 2| Edit the SQL statement, or choose from the menu.

Each SQL statement you create with UseSQL contains the following information:

- Created By: The name of the user who created the statement.
- Creation Date: The date on which the statement was created.
- Last Run: The date on which the statement was last executed. If the statement has not been executed, this field is blank.
- Connection: The connection for which this statement was originally created. The connection can be any valid SQL Link connection (including a custom connection).

Renaming a UseSQL card

To change the title of this card, choose Edit | RenameCard. Type the new name in the Title text box.



```
SELECT DISTINCT CUSTOMER NO, CITY, STATE
FROM JLEE.CUSTOMER
WHERE
        (ZIP CODE
ORDER BY CUSTOMER
```

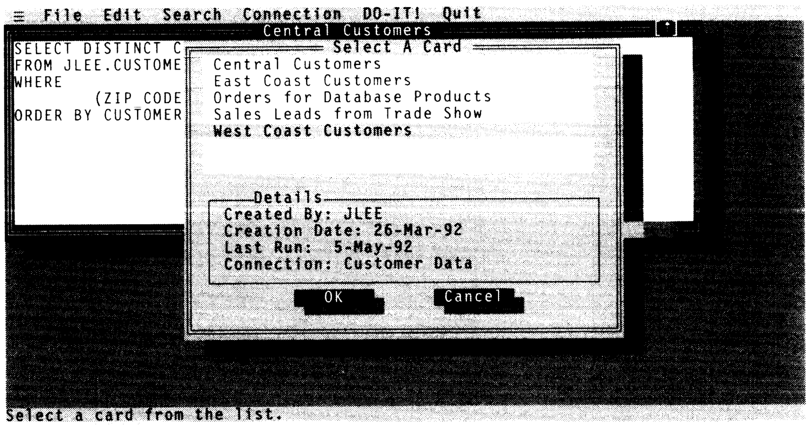
Title: Central Customers

Details
Created By: JLEE
Creation Date: 1-May-92
Last Run: 5-May-92
Connection: Customer Data

Change the title of the current SQL statement.

Navigating between cards

To access another SQL statement you can use File | Open from the UseSQL Main menu.



```
SELECT DISTINCT C
FROM JLEE.CUSTOME
WHERE
        (ZIP CODE
ORDER BY CUSTOMER
```

Central Customers
East Coast Customers
Orders for Database Products
Sales Leads from Trade Show
West Coast Customers

Details
Created By: JLEE
Creation Date: 26-Mar-92
Last Run: 5-May-92
Connection: Customer Data

Select a card from the list.

You can also move between cards with the keyboard. Press *Ctrl-PgUp* or *Ctrl-PgDn* to display the previous or next UseSQL card, respectively. To go to the first card in the stack, press *Ctrl-Home*. To go to the last card, press *Ctrl-End*.

Editing SQL statements

You can modify your commands using the editing keys listed in Table 4-3. You can also use your mouse to move around a UseSQL card, select text, and choose menu options. For additional information on features available in the Editor, see Chapter 12 of the Paradox *User's Guide*.

Table 4-3 Table editing keys

Key	Description
Editing keys	
Ins	Toggles between insert and overwrite mode.
Del	Deletes the current character if no area is highlighted or erases entire highlighted area.
Ctrl-Ins	Copies the selected (highlighted) area to the Clipboard. (This replaces any text that was previously copied to the Clipboard.)
Shift-Del	Cuts the selected (highlighted) area to the Clipboard. (This replaces any text that was previously copied to the Clipboard.)
Shift-Ins	Inserts (pastes) the contents of the Clipboard at the current cursor position.
Backspace	Deletes one character to left of cursor.
Navigation on the same card	
←	Moves cursor one character left.
→	Moves cursor one character right.
↑	Moves cursor to previous line.
↓	Moves cursor to next line.
Home	Moves cursor to start of the current line.
End	Moves cursor to end of current line.
Ctrl ←	Moves cursor to end of previous word.
Ctrl →	Moves cursor to beginning of next word.
PgUp	Moves cursor up one screenful in the current SQL statement.
PgDn	Moves cursor down one screenful in the current SQL statement.
Navigation between cards	
Ctrl-Home	Moves cursor to start of first command in the stack.
Ctrl-End	Moves cursor to start of last command in the stack.
Ctrl-PgUp	Moves cursor to start of previous statement.
Ctrl-PgDn	Moves cursor to start of next statement.

Edit menu

In addition to these editing keys, you can use UseSQL's Edit menu to perform operations on blocks of text and UseSQL cards. The following table describes these options.

Table 4-4 Edit menu options

Menu option	Description
XCut	Cuts the selected text to the Clipboard.
Copy	Copies the selected text to the Clipboard.
Paste	Inserts the contents of the Clipboard at the location of the cursor.
Erase	Erases the selected text in the current SQL statement and does not store it on the Clipboard.
Goto	Lets you specify a particular line to go to.
Location	Displays the line and column position of your cursor in the current SQL statement
ShowClipboard	Shows you the contents of the Clipboard.
RenameCard	Changes the title of the current UseSQL card. You can use any descriptive word or phrase, up to 35 characters.
DeleteCard	Deletes the current UseSQL card. When you do so, the next UseSQL card appears.

Search menu

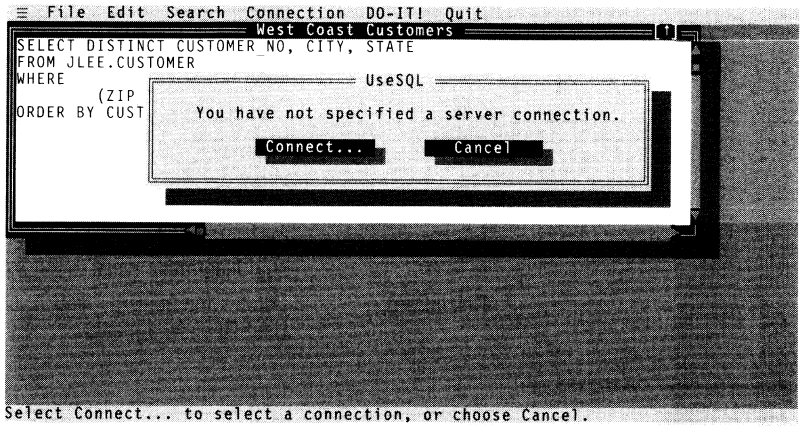
UseSQL lets you find and replace strings in SQL statements using the Search menu. The following table describes the options available from this menu.

Table 4-5 Search menu option

Menu option	Description
Find	Finds a text string in the current SQL statement.
Next	Finds the next occurrence of the last-defined text string in the current SQL statement.
Replace	Performs a single search-and-replace operation in the current SQL statement. Brings up dialog boxes in which you can define the strings to search for and to replace.
ChangeToEnd	Replaces all instances of a specified string in the current SQL statement. Brings up a dialog box in which you can define the strings to search for and to replace.

Executing a SQL statement

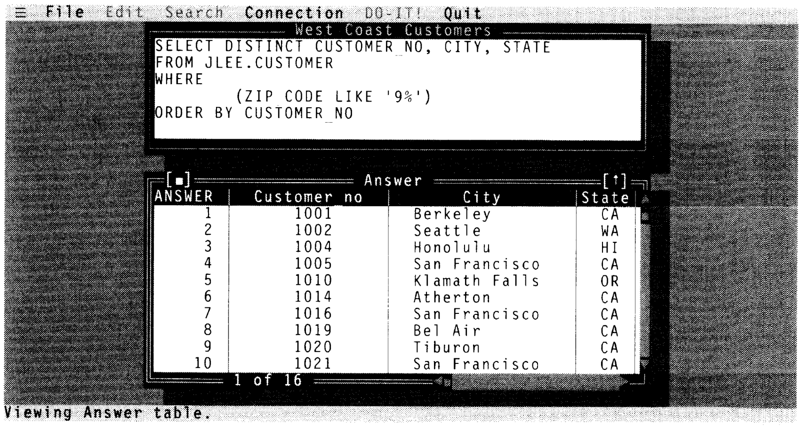
When you're ready to test your SQL statement, press *F2* Do_It!. Paradox sends your statement to the server for processing. If you are not connected to a server, UseSQL gives you the option to connect.



```
≡ File  Edit  Search  Connection  DO-IT!  Quit
                  West Coast Customers
SELECT DISTINCT CUSTOMER_NO, CITY, STATE
FROM JLEE.CUSTOMER
WHERE                        UseSQL
      (ZIP
ORDER BY CUST    You have not specified a server connection.

                      Connect...         Cancel

Select Connect... to select a connection, or choose Cancel.
```

If your command is a query (for example, a SELECT statement) and the database server produces a result, UseSQL captures the result in a local Paradox *Answer* table and displays it.

With an *Answer* table selected, you see the following menu (some menu options are unavailable as they don't apply to tables):



```
≡ File  Edit  Search  Connection  DO-IT!  Quit
                  West Coast Customers
SELECT DISTINCT CUSTOMER_NO, CITY, STATE
FROM JLEE.CUSTOMER
WHERE
        (ZIP_CODE LIKE '9%')
ORDER BY CUSTOMER_NO

[■]                        Answer                      [↑]
ANSWER |  Customer no  |       City        |State
     1 |     1001      | Berkeley          |  CA
     2 |     1002      | Seattle           |  WA
     3 |     1004      | Honolulu          |  HI
     4 |     1005      | San Francisco     |  CA
     5 |     1010      | Klamath Falls     |  OR
     6 |     1014      | Atherton          |  CA
     7 |     1016      | San Francisco     |  CA
     8 |     1019      | Bel Air           |  CA
     9 |     1020      | Tiburon           |  CA
    10 |     1021      | San Francisco     |  CA
          1 of 16
Viewing Answer table.
```

When the *Answer* table is selected, you can use File | CopyTo to save it under another name. If you don't save the *Answer* table, Paradox will delete it (as it deletes other temporary tables) when you leave Paradox.

When you select the UseSQL card, all of the menu selections become available again.

Some SQL statements, such as a DROP TABLE statement, don't produce an *Answer* table. In these cases, UseSQL displays a summary of your statement's results in the lower right message area.

SQL Link displays any messages produced by your server or by Paradox. If your command produces an error on the server, SQL Link displays the resulting error message. An error might result from a failure in the server connection or a problem in your SQL statement. You'll need to correct the error before you can execute the statement successfully. Refer to Table 4-3 for a list of editing keys and their functions.

Leaving UseSQL

Choose Quit to leave UseSQL and return to the Paradox Main menu. UseSQL automatically saves your changes.

Learning more about SQL

To learn more about SQL programming, refer to your database server manuals, where your server's specific implementation of SQL statements is fully explained. To learn more about using PAL's SQL commands, see Chapter 6.

CHAPTER 5

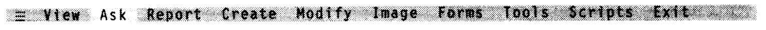
Menus

This chapter describes the menu commands SQL Link adds to Paradox. It also describes how some of the familiar Paradox menu commands work with SQL Link.

Only those commands that are unique to SQL Link or that work differently when you're using SQL Link are discussed in this chapter. If you try to use a Paradox menu command that is not valid for remote operations, you'll get an error message. Keep in mind that you can always query a remote table then create a local table by renaming the resulting *Answer* table, and perform any Paradox operation on the local table. If you need information about a menu command that is not included here, see the Paradox *User's Guide*.

This chapter presents the menu commands for SQL Link in the order in which they appear on the Paradox Main menu.

Ask


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit

You can query data in remote tables, using Ask, in much the same way you query local Paradox tables. To query a remote table,

1. Choose Ask from the Main menu. Paradox prompts you for a table name.
2. Type the name of the remote table you want to work with, or choose it from the list box. Paradox prompts you for your user name and password if necessary.

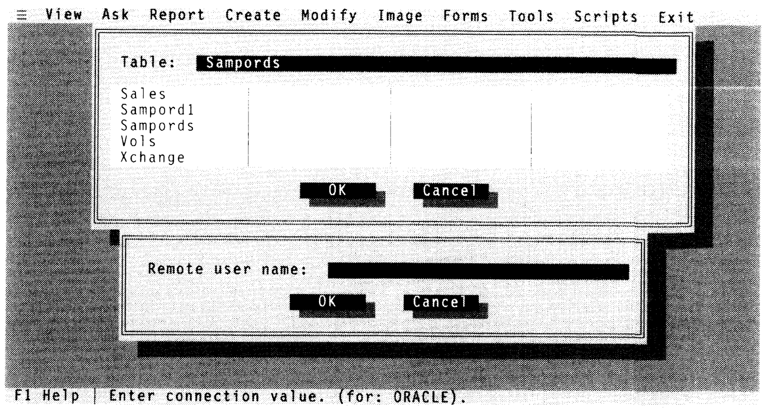

≡ View Ask Report Create Modify Image Forms Tools Scripts Exit

Table: Sampords

Sales
Sampord1
Sampords
Vols
Xchange

Remote user name:

F1 Help | Enter connection value. (for: ORACLE).

3. Type your remote user name and press *Enter.* Paradox prompts you for your remote password.

4. Type your remote password and press *Enter.* Paradox displays a *Query* form for the table you've selected.

5. Fill in the *Query* form as usual and press *F2* Do_It!.

Note Not all Paradox *Query* operators are valid in the SQL environment. See Table 5-2 for details.

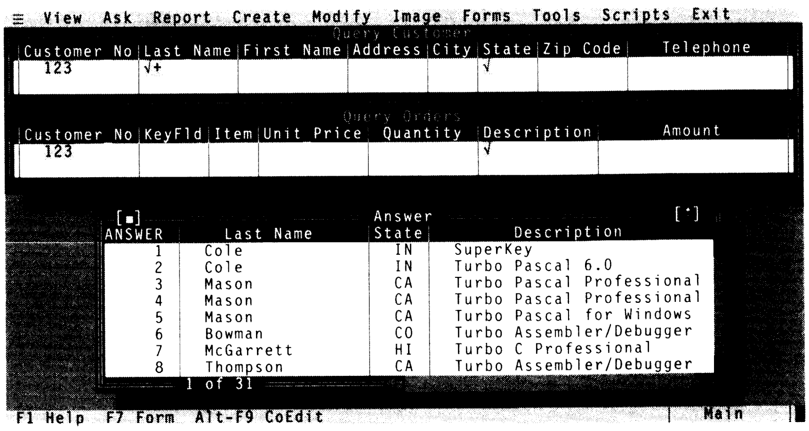
Paradox *validates* your query, translates it into the appropriate SQL dialect, sends it to the server, and displays the result in a local *Answer* table. You can treat this *Answer* table as you would any *Answer* table. For example, you could rename it (using Tools | Rename), and then use it to generate forms, reports, and graphs.

For more information about queries, see Chapter 5 in the Paradox *User's Guide.*

Multiple Query forms

SQL Link lets you have more than one *Query* form on the workspace. You can combine queries on several remote tables, provided all remote tables on the workspace use the same SQL connection.

You can't combine local and remote tables in a single query, nor can you combine remote tables from different connections. You can, however, query remote tables and rename the resulting *Answer* tables as local tables. You can then combine these local tables in any valid Paradox query. The following is an example of a *join* between two remote tables.



```
≡ View  Ask  Report  Create  Modify  Image  Forms  Tools  Scripts  Exit
                              Query Customer
 Customer No|Last Name|First Name|Address|City|State|Zip Code|  Telephone
   123      |√+       |          |       |    |√    |        |

                              Query Orders
 Customer No|KeyFld|Item|Unit Price| Quantity |Description|      Amount
   123      |      |    |          |          |√          |

        [■]                        Answer                              [↑]
       ANSWER      Last Name       State        Description
           1    Cole               IN     SuperKey
           2    Cole               IN     Turbo Pascal 6.0
           3    Mason              CA     Turbo Pascal Professional
           4    Mason              CA     Turbo Pascal Professional
           5    Mason              CA     Turbo Pascal for Windows
           6    Bowman             CO     Turbo Assembler/Debugger
           7    McGarrett          HI     Turbo C Professional
           8    Thompson           CA     Turbo Assembler/Debugger
            1 of 31
 F1 Help  F7 Form  Alt-F9 CoEdit                                  Main
```

Be aware that the data in the *Answer* tables is not "dynamic"; it is a snapshot of the contents of the remote table and is not updated when the data on that table changes.

SQL query operators

SQL Link lets you construct remote table queries using all Paradox reserved words and operators listed in Table 5-1 and *none* of those listed in Table 5-2. See Chapter 5 of the Paradox *User's Guide* for explanations and examples of how to use operators in your queries.

Table 5-1 Supported SQL Link QBE operators

Category	Operator	Meaning
Reserved words and symbols	✓	Display field in *Answer*
	✓+	Display field and include duplicate values
	✓▼	Display field with values in descending order
	CALC	Calculate new field
	INSERT	Insert new records with specified values
	DELETE	Remove selected records from table
	CHANGETO	Change values in selected records
Arithmetic operators	+	Addition or concatenation
	–	Subtraction
	*	Multiplication
	/	Division

Category	Operator	Meaning
	()	Group operators in a query expression
Comparison operators	=	Equal to (optional)
	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less than or equal to
Wildcard operators *	..	Any characters
	@	Any single character
Special operators	NOT	Does not match
	BLANK	No value
	TODAY	Today's date
	OR	Specify OR conditions in a field
	,	Specify AND conditions in a field
	AS	Specify name of field in *Answer*
Summary operators	AVERAGE	Average of values
	COUNT	Number of values
	MAX	Highest value
	MIN	Lowest value
	SUM	Total of the values
	ALL	Calculate summary based on all values in group, including duplicates

* On case-sensitive servers, the wildcard operators @ and .. always produce a case-insensitive search pattern.

Table 5-2 Paradox QBE operators not supported on remote tables

Category	Operator
Reserved words	G FIND SET FAST
Special operators	LIKE !
Set comparison operators	ONLY NO EVERY EXACTLY

In addition, the following restrictions apply:

- While you can use summary operators with the CALC operator (for example, CALC SUM), you cannot use restricted aggregation or aggregate constants in remote queries (for example, AVERAGE <17, SUM X).
- You cannot use multiline or multi-table DELETE or CHANGETO queries.
- You cannot use wildcard (pattern) operators in date or numeric fields.

Transaction processing and queries

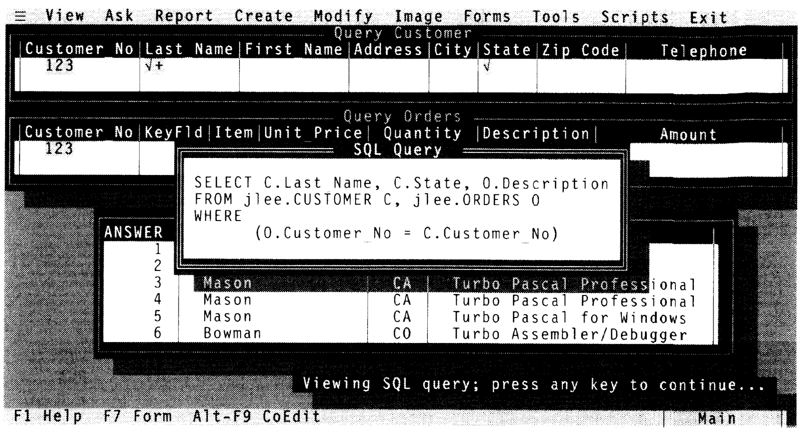
If you're building an application using transaction processing, the AutoCommit setting affects record locking after queries; AutoCommit Yes performs a COMMIT after each query. See the discussion of AutoCommit, later in this chapter, and your server-specific addendum for details.

Displaying your SQL query

SQL Link automatically translates your query into a SQL statement that your database server understands.

Alt-F2 **ShowSQL**

Whenever you have a query on the workspace, you can press *Alt-F2* ShowSQL to display the SQL translation of the query. When you press *Alt-F2* ShowSQL, Paradox displays the SQL statement in a box on the screen, as illustrated in the following figure.



```
≡ View  Ask  Report  Create  Modify  Image  Forms  Tools  Scripts  Exit
                         Query Customer
|Customer No|Last Name|First Name|Address|City|State|Zip Code|   Telephone  |
   123       √+                                   √

                         Query Orders
|Customer No|KeyFld|Item|Unit Price| Quantity |Description|        Amount  |
   123                      SQL Query

            SELECT C.Last Name, C.State, O.Description
            FROM jlee.CUSTOMER C, jlee.ORDERS O
            WHERE
 ANSWER          (O.Customer_No = C.Customer_No)
      1
      2
      3   Mason                  CA   Turbo Pascal Professional
      4   Mason                  CA   Turbo Pascal Professional
      5   Mason                  CA   Turbo Pascal for Windows
      6   Bowman                 CO   Turbo Assembler/Debugger

                         Viewing SQL query; press any key to continue...
 F1 Help  F7 Form  Alt-F9 CoEdit                                   | Main
```

When you're through viewing the query, press any key or click a mouse button to clear the box and return to the workspace.

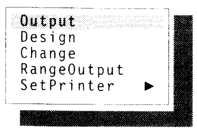
Saving your query

Just like a query form for a local Paradox table, you can save a remote query in a PAL script. With SQL Link, you have two options:

- You can save it in a PAL QUERY...ENDQUERY command by choosing the Scripts | QuerySave command. For more information, see the discussion of QuerySave in Chapter 18 of the Paradox *User's Guide*.
- You can save it as a SQL statement enclosed in the PAL SQL...ENDSQL command by choosing the Tools | SQL | SQLSave command. For more information, see the discussion of Tools | SQL | SQLSave later in this chapter.

In either case, you can repeat your query by playing the script by itself or including it in a PAL application.

Report



Choose Report from the Main menu to generate a report from data in remote tables. You can print a report of data in any remote table using standard or custom reports.

You can use all the Report commands with remote tables just as you would with local Paradox tables:

Output	Prints a report to a printer, the screen, or a file.
Design	Lets you create a custom report for the remote table.
Change	Lets you change a report specification.
RangeOutput	Prints selected pages of a report.
SetPrinter	Chooses and sets up the default printer.

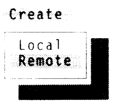
You cannot design multi-table reports on remote tables. You can, however, query several remote tables, and combine the local *Answer* tables with other local tables to create a multi-table report.

Data derived from remote tables will not reflect subsequent changes made to the original tables. If you suspect that data on the remote table changes frequently, be sure to run reports promptly after creating your *Answer* tables(s).

If you are reporting on only a portion of the data in a large remote table, it's probably more efficient to query the table (using Ask), and extract only those fields and records you need. Simply rename the resulting *Answer* table to a local Paradox table, then run your report from this table. You can also choose Tools | Copy | JustFamily to copy custom reports and forms from the remote table's replica to this new table (provided the structure of the *Answer* table and replica match in terms of fields' number, order, and type).

For more information about reports, see Chapter 7 of the Paradox *User's Guide*.

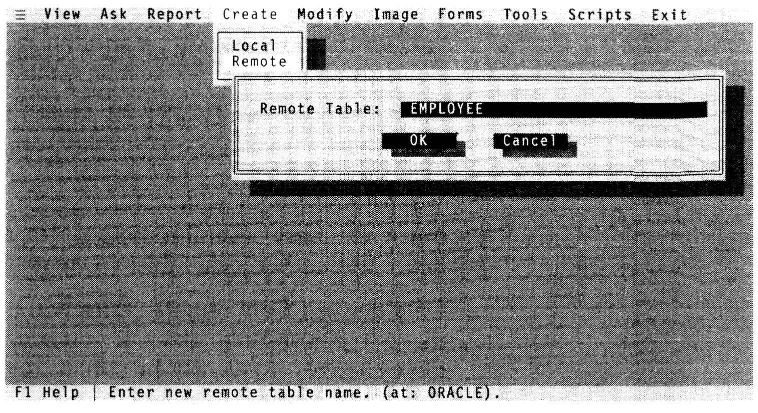
Create



Choose Create from the Main menu to create a new remote table. You must select a connection before you can create a remote table. You can do this using the Tools | SQL | Connection | Select command. For more information, see the discussion of Select later in this chapter.

To create a remote table,

1. Choose Create from the Main menu. If you've selected a server connection, Paradox asks you whether you're creating a local table or a remote table.
2. Choose Remote to create a remote table.
3. Type a name for the new table and press *Enter*. The name should conform to the naming rules for Paradox tables and your database server's rules. Paradox warns you if a table with this name already exists or displays an error if the table name is invalid. If this happens, choose a new name for the table.


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Local
Remote
OK
Cancel

Remote Table: EMPLOYEE

F1 Help | Enter new remote table name. (at: ORACLE).

Paradox displays the name of the current server connection to remind you where it will create the new table. Paradox also displays the table name you type in uppercase letters to remind you that the remote table will be created with all uppercase letters on the server. If the table name is valid, Paradox displays a *structure image.*

4. Define the structure of the remote table by filling out the structure image as you would for a local table. You can use all the standard Paradox field types (**N**, **$**, **S**, **A**, and **D**) except memo (**M**) and binary (**B**). If a field type is not supported on your server, Paradox maps to the closest type that is supported.

   - Field names must comply with the rules of the database server you are using. Servers do not allow spaces, unusual characters (such as hyphens), or reserved words (such as SELECT, KEY, DATE, DESC, or NUMBER) in field names. Paradox will not let you type a space. If you use a reserved word, Paradox will return an error message when you press *F2* Do_It! to Create. If you need more information about field-naming restrictions for your particular server, see your server-specific addendum or your server manuals.

   - You can create an index for the remote table by defining a primary key as you would for a local Paradox table. Any index you create must conform to both Paradox and your server's rules. Once you create a remote table with a primary index, your database server automatically maintains the index if other users update the table you have created. For more information about indexes on remote tables, see "Paradox/server differences" in Chapter 2 of this manual.

Memo fields

- You cannot use SQL Link to create memo fields on remote tables. If you attempt to define a field as field type **M**, you get an error message. If your server supports memo fields, when you view data from these fields in existing remote tables it appears as type A255.

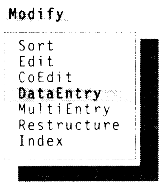
5. When you're finished, press *F2* Do_It!. Paradox creates the remote table and its replica. Paradox also creates a primary index and a replica index, if you've defined an index.

If Paradox is unable to create the table, it displays an error message. Verify that the field names in the table are permitted by your database server.

Paradox always automatically issues a COMMIT before and after a Create operation, so you cannot roll it back. If AutoCommit is set to No, be aware that using Create will commit any pending transactions. See "Transaction processing" in Chapter 6 for more information.

For more information about using Create, see Chapter 10 of the Paradox *User's Guide*.

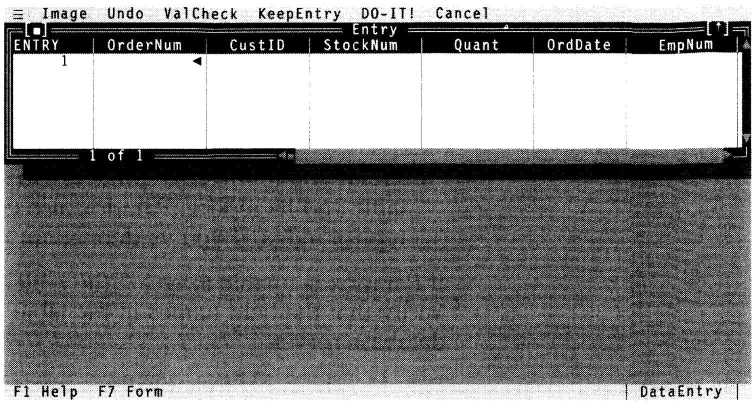
Modify | DataEntry



You can enter data in a remote table using Modify | DataEntry.

To add data to a remote table,

1. Choose Modify | DataEntry.
2. Type the replica name of the table you want to add data to, or choose it from the list box.


≡ Image Undo ValCheck KeepEntry DO-IT! Cancel
Entry
ENTRY | OrderNum | CustID | StockNum | Quant | OrdDate | EmpNum
1
1 of 1
F1 Help F7 Form
DataEntry

Note SQL Link does not let you view or edit remote SQL tables directly.

Paradox displays an empty table called *Entry*. This table has the same structure as the table you want to add records to.

If you want to enter data using a data entry form (such as a custom form you created using Forms | Design, described later in this chapter), press *F7* Form Toggle. For more information about selecting custom forms, see the discussion of Modify | DataEntry in Chapter 11 of the Paradox *User's Guide.*

You can also create *picture strings* and *validity checks*. Press *F10* Menu to access the appropriate menu for these operations.

3. Enter the records you want to add to the remote table.
4. When you're finished, verify that the data is correct, then press *F2* Do_It!.
   - ❐ If the remote table has an index and key violations occur, Paradox puts the duplicate records in a local *Keyviol* table. See the discussion of key violations in Chapter 11 of the Paradox *User's Guide.*
   - ❐ If other problems occur (such as adding NULL values to fields in a remote table that do not allow nulls, or some other data integrity conflict), Paradox generates a *Problems* table. For more information, see the discussion of *Problems* tables in Chapter 17 of the Paradox *User's Guide.*
   - ❐ If the server connection fails, or you interrupt the process with *Ctrl-Break*, you can save the records you entered in a local Paradox table by pressing *F10* Menu and choosing DataEntry | KeepEntry. For more information, see the discussion of KeepEntry in Chapter 11 of the Paradox *User's Guide.*

How to restructure a remote table

Paradox does not support Modify | Restructure for remote tables. If you need to restructure a remote table, you can create the new structure, and then use an INSERT query to add selected data.

If you change a remote table's structure with SQL statements, run SQL Setup to create a new Paradox replica, then run an INSERT or UPDATE query to fill in the data.

Validity checks

Validity checks can be placed on an *Entry* table using Paradox picture rules. To keep validity checks with the replica, place these first on the *Entry* table. Since *Entry* will be deleted when you exit Paradox, use Tools | Copy | JustFamily to copy the family members of the *Entry* table to the replica and keep those family members permanent.

Forms

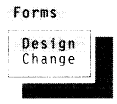
You can use forms for a replica. When you choose Modify | DataEntry, all family members (including forms and validity checks) are copied to the DataEntry table.

Nulls

When SQL Link creates remote tables, it creates columns that allow nulls. If you don't want the remote table to allow nulls, create the table using passthrough SQL and replicate it using SQL Setup.

For more information on DataEntry, see Chapter 11 of the Paradox *User's Guide.*

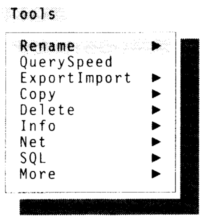
Forms



Choose Forms from the Main menu to create data entry forms. For more information about forms, see Chapter 15 of the Paradox *User's Guide.*

You can design or change a form, but you cannot directly create multi-table forms for remote tables. You can, however, query remote tables, rename the *Answer* tables, and create multi-table forms for the local tables that contain snapshots of remote data.

Tools



The Tools command from the Main menu lets you manage the Paradox environment.

You can use the following commands from the Tools menu with remote tables (other commands on this menu don't apply to remote tables):

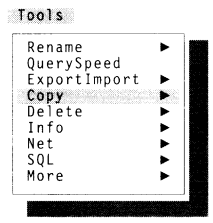
Copy	Copies local or remote tables.
Delete	Deletes local or remote tables.
SQL	Manages your session in the SQL environment.
More	Displays another menu with commands that let you copy and delete records, and password-protect replicas.

These tools perform operations on both local and remote tables. To work directly with replicas, see the discussion of SQL | ReplicaTools later in this chapter.

Note If the SQL command does not appear on the Tools menu, SQL Link is either not installed or not activated. See your server-specific addendum for more information.

Paradox looks for a PARADOX.OV3 file to determine if SQL is installed. You need to have the PARADOX.OV3 file present in the Paradox system files directory to enable SQL Link.

Copy



Tools | Copy lets you copy data from an existing table to a new table. Copy lets you copy

❒ a local table to a local table

❒ a local table to a remote table

❒ a remote table to a local table

❒ a remote table to a remote table, even from one server to another server

If you are copying data to a remote table, the source (local) table structure must comply with the server's field-naming rules. This means that neither table can include embedded spaces in or use reserved words for field names. In addition, the data-type boundaries (such as dates and numbers) must comply with the server's rules.

Before you copy data to a remote table, you need to tell Paradox where to put the new table. To do this, use Tools | SQL | Connection | Select to choose a connection. For more information, see the discussion of Tools | SQL | Connection | Select later in this chapter.

To copy a table,

1. Choose Tools | Copy | Table from the Main menu.

2. Type the name of the source (local) table you want to copy, or choose it from the list box. (Paradox prompts you for your remote user name and password if necessary.)

   Paradox then asks you whether the new table will be local or remote.

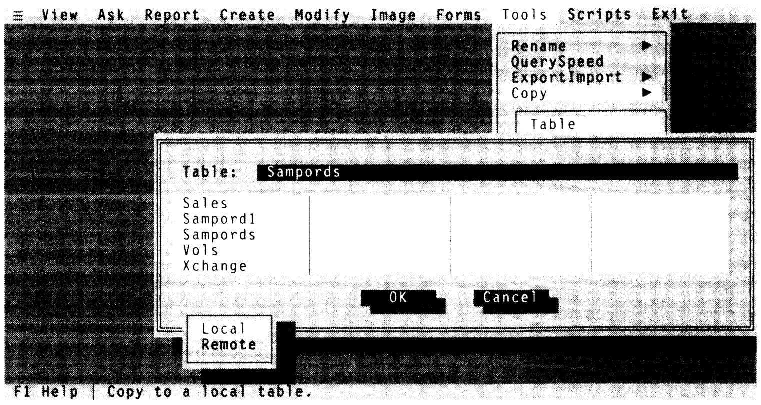

≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Rename ►
QuerySpeed
ExportImport ►
Copy ►
Table
Local
Remote
OK
Cancel

Table: Sampords

Sales
Sampordl
Sampords
Vols
Xchange

F1 Help | Copy to a local table.

3. Choose Remote to create a new table on the database server specified by your current connection. (Local creates a new Paradox table on your local drive.)

4. Type the name of the target (remote) table. To protect you from accidentally overwriting an existing table, Paradox warns if the name you enter already exists on the database server or in the current working directory. If this happens, and you don't want to overwrite the existing table, type a different name.

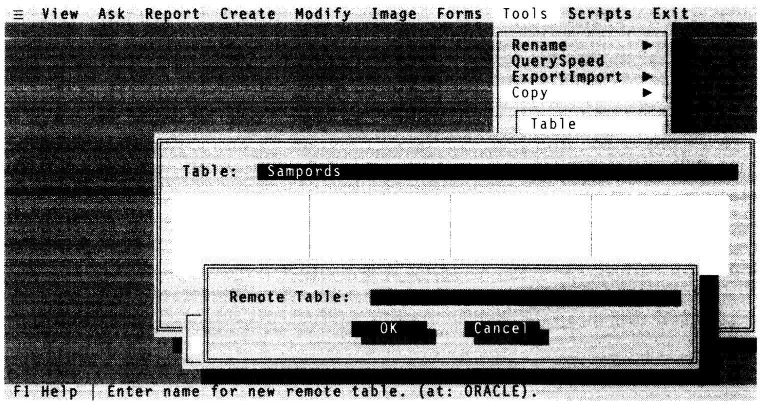

≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Rename ►
QuerySpeed
ExportImport ►
Copy ►
Table
OK
Cancel

Table: Sampords

Remote Table:

F1 Help | Enter name for new remote table. (at: ORACLE).

Paradox creates a copy of the table. If the target is a remote table, Paradox also creates its replica.

- If there is a primary index for the source table, Paradox automatically creates an index for the target table as well.
- If for some reason the server connection fails, or you interrupt the process with *Ctrl-Break*, the target table is not created.

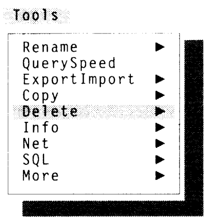
Paradox automatically issues a COMMIT before and after the copy operation, so you cannot roll it back. Be aware that Copy will commit any pending transactions, even when AutoCommit is set to No. See "Transaction processing" in Chapter 6 for more information.

Copy is a single operation.

Paradox treats Copy as a single, atomic operation. If the operation fails, the entire operation is aborted. Copy will fail if the source (local) table contains data that is invalid for the target (remote) table. (For example, the source table could contain a date that is valid in Paradox, but not on the database server.) If Copy fails, create a new target table, borrow the structure from the source table, and add the records from the source table to the target table. If an invalid record is detected, it is stored in a *Problems* table, and the ADD proceeds.

For more information about Copy, see Chapter 17 of the Paradox *User's Guide.*

Delete



Use Tools | Delete | Table to delete a remote table.

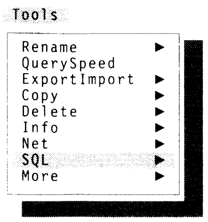
1. Choose Tools | Delete | Table from the Main menu.
2. Type the name of the remote table you want to delete, or choose it from the list box.
3. To protect you from accidentally deleting a table, Paradox shows you the fully-qualified table name that the replica refers to, and asks you to confirm that you really want to delete it. Choose Cancel to stop or OK to continue.

Paradox directs the server to delete the remote table, along with any associated remote indexes. Paradox also deletes the replica and any associated local objects such as forms or reports.

Paradox automatically issues a COMMIT before and after the Delete operation, so you cannot roll it back. See "Transaction processing" in Chapter 6 of this manual for more information.

For more information about Delete, see Chapter 17 of the Paradox *User's Guide.* Also see Table 5-3 later in this chapter for more on the difference between Empty and Delete operations on remote tables.

SQL

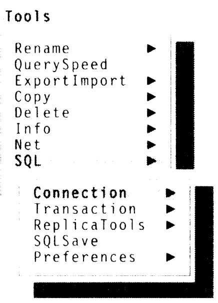


Use Tools | SQL to manage your work with remote tables.

When you choose Tools | SQL you see the SQL menu with the following commands:

Connection	Lets you manage your server connection.
Transaction	Lets you begin a transaction on the server and commit or roll back changes to remote tables.
ReplicaTools	Lets you copy, delete, or rename replicas.
SQLSave	Saves the SQL translation of a query to a PAL script.
Preferences	Lets you fine-tune SQL environment settings.

Connection

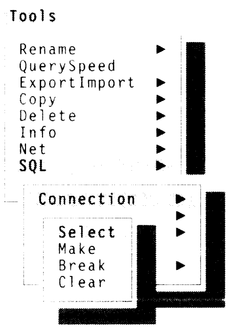


Use Tools | SQL | Connection to select and manage a connection with a database server.

When you choose Tools | SQL | Connection, you see the following commands:

Select	Lets you select a server connection from a list of available connections and connects to the selected server.
Make	Lets you re-establish a server connection.
Break	Disconnects from the server.
Clear	Clears the current connection, the workspace, and all remote user names and passwords.

Select



Use Tools | SQL | Connection | Select, to select a server from a list of available connections and immediately connect to that server.

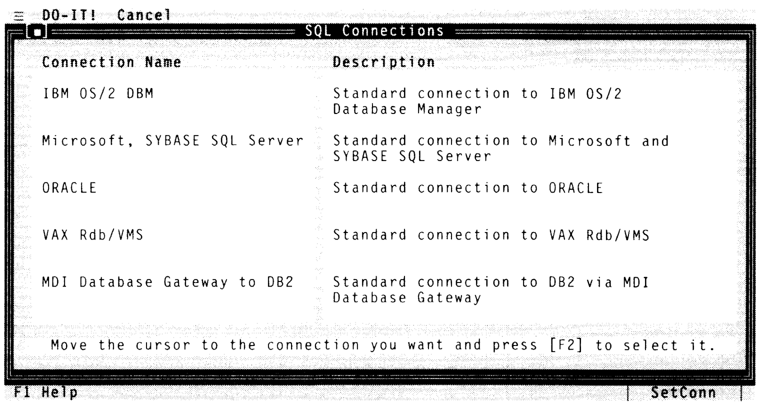
Paradox uses this connection to identify the location of any new remote tables you create with Create | Remote or Tools | Copy | Remote, and to direct any SQL commands you issue from PAL. When you choose Select, Paradox displays the list of available connections.

If you have already used a replica in your current Paradox session, you do not have to explicitly connect to a server if the replica connection is already established.

Use the SQL Setup Program to customize your list of connections. For more information, see Chapter 7 of this manual.

To select a connection,

1. Choose Tools | SQL | Connection | Select. You'll see a screen listing the available server connections (the server products that you've installed or customized):



```
≡ DO-IT!  Cancel
                         SQL Connections
 Connection Name                 Description

 IBM OS/2 DBM                    Standard connection to IBM OS/2
                                 Database Manager

 Microsoft, SYBASE SQL Server    Standard connection to Microsoft and
                                 SYBASE SQL Server

 ORACLE                          Standard connection to ORACLE

 VAX Rdb/VMS                     Standard connection to VAX Rdb/VMS

 MDI Database Gateway to DB2     Standard connection to DB2 via MDI
                                 Database Gateway

  Move the cursor to the connection you want and press [F2] to select it.

F1 Help                                                    | SetConn |
```

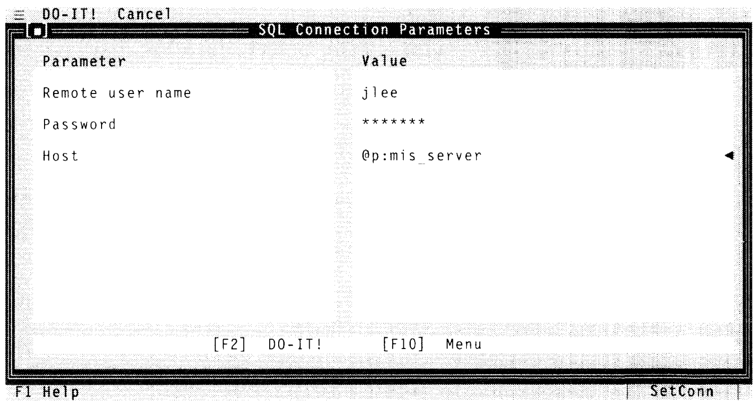
You can change the connection names and descriptions using SQL Setup. For more information, see Chapter 7.

2. Select the server connection you want to use, then press *F2* Do_It!. If you don't want to connect, press *F10* Menu and choose Cancel.

   When you press *F10* Menu, you'll see a menu with the following commands:

DO-IT!	Selects the current connection, just as if you'd pressed *F2* Do_It!.
Cancel	Lets you cancel the operation and returns you to the Connection menu.
Help	Displays help about making connections.

3. You might need to supply additional information (such as user name, password, or server name) to Paradox to define a particular server connection. If so, you'll see a second screen prompting you for the parameters you need to supply:



```
≡ DO-IT! Cancel
SQL Connection Parameters
Parameter                          Value
Remote user name                   jlee
Password                           *******
Host                               @p:mis_server

              [F2] DO-IT!      [F10] Menu
F1 Help                                        SetConn
```

   You can provide default values for these parameters using SQL Setup.

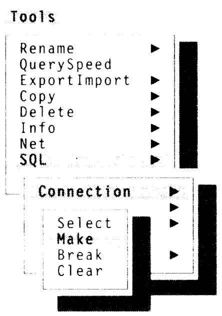
   Use the mouse or ↑ and ↓ keys to move from field to field, and type the necessary information. When you're finished, press *F2* Do_It! to establish the connection. If you want to cancel your selection, press *F10* Menu to activate the SQL menu and choose Cancel. You can also press *Esc* to return to the previous screen, and choose another connection instead.

   For more information on the connection parameters required by your server, see your server-specific addendum and your server manuals. If you leave an optional parameter blank, Paradox uses the server's default value. If you leave a required parameter blank, Paradox will not be able to proceed until you provide a value.

If you don't have the appropriate communications drivers loaded, Paradox won't be able to connect and gives you an error message. If this happens, you must exit to DOS and load the drivers, then restart Paradox.

Important Don't shell to DOS with *Alt-0* DOSBig or the PAL DOSBIG or RUN BIG commands; these drivers are TSRs, and may cause memory conflicts when you try to return to Paradox.

Make



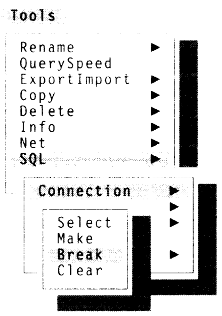
Tools | SQL | Connection | Make tells Paradox to reconnect to a server using the current connection information.

Use this command after you break a connection in one of the following ways:

- ❒ by selecting a different server connection
- ❒ by choosing Tools | SQL | Connection | Break
- ❒ by pressing *Ctrl-Break* during a remote operation
- ❒ by executing the PAL DOSBIG or RUN BIG commands
- ❒ by using *Alt-0* DOSBig
- ❒ by exiting Paradox

If you haven't yet selected a server connection and you choose Make, you'll get an error message.

Break



Tools | SQL | Connection | Break lets you disconnect from the current database server.

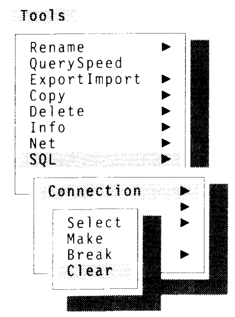
When you choose Break, Paradox rolls back the current transaction (if any) on the server, closes remote files, and disconnects from the server. By choosing Break to disconnect when you no longer need to be connected to a server (for example, if you won't need access to remote tables for a while), you free up server resources.

You can also break the current server connection in the following ways:

- ❒ by selecting a different server connection
- ❒ by pressing *Ctrl-Break* during a remote operation
- ❒ by executing the PAL DOSBIG or RUN BIG commands
- ❒ by using *Alt-0* DOSBig
- ❒ by exiting Paradox

Important When AutoCommit is set to No and you break the connection to the server, Paradox does not commit the changes on that server, and the server rolls back any open transaction.

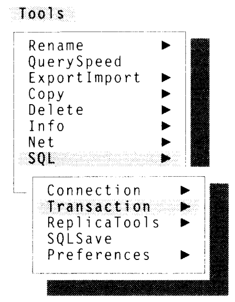
Clear



Tools | SQL | Connection | Clear clears the current server connection, the workspace, and all remote user names and passwords for the current connection.

When you clear the server connection, Paradox acts as if SQL Link is not running. Until you select another server connection, you won't see the option to create a remote table when you choose Create.

Transaction



The Transaction menu lets you explicitly begin a transaction and commit or roll back a transaction on a database server.

A transaction can be a single operation (like adding records from one table to another), or a series of operations (like adding records from one table to another, then deleting the records in the source table once the records are added successfully). To conclude a transaction, you either commit (save) or roll back (abandon) your changes to remote data.

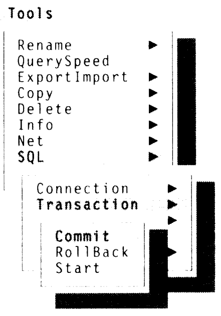
Paradox automatically commits certain operations for you (the data-definition commands Create, Tools | Copy, and Tools | Delete, and their equivalent PAL commands). Therefore, you cannot roll back these operations, regardless of the setting of AutoCommit.

When you choose Tools | SQL | Transaction, you see a menu with these options:

Commit	Saves changes you've made to remote tables.
RollBack	Abandons changes to remote tables if you decide not to save them (for example, if an error occurred while updating several tables).
Start	Begins a transaction on the database server. If your server does not automatically begin transactions, you must use Start if you want to be able to roll back your changes.

Whether you need to issue an explicit COMMIT or ROLLBACK on your server depends upon the setting of AutoCommit. For example, you always need to commit or roll back changes if AutoCommit is set to No or if you're using the PAL SQL...ENDSQL command. For more information, see the discussion of AutoCommit later in this chapter and the discussion of SQL...ENDSQL in Chapter 6.

Commit

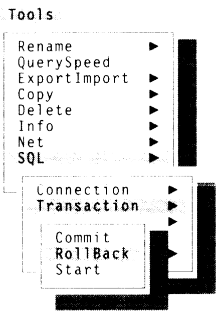


Tools | SQL | Transaction | Commit lets you save changes on the database server.

When AutoCommit is set to Yes, Paradox automatically commits changes to remote tables at the conclusion of every menu operation. If you set AutoCommit to No, however, Paradox saves your changes only when you choose Commit. Remember that Paradox automatically commits certain operations, regardless of the AutoCommit setting. Commands executed with the SQL...ENDSQL command are not affected by the setting of AutoCommit. Paradox never commits these operations, so you must explicitly commit them (if your server does not). For more information, see the discussion of AutoCommit later in this chapter.

Commit succeeds even if there are no changes to save. Commit fails if you're not connected to a server (on some servers, Commit fails if there is no transaction pending).

RollBack



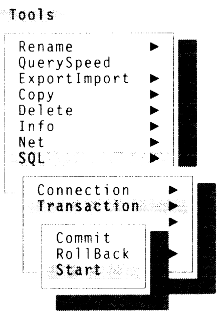
Tools | SQL | Transaction | RollBack lets you cancel changes made to remote tables and restore the remote data to the state it was before you began the transaction.

You might want to roll back changes to remote tables if an operation failed during a transaction.

If your server does not automatically begin transactions, you must use Start if you want to be able to roll back your changes.

RollBack succeeds even when there are no changes to undo. RollBack fails if you're not connected to a server (on some servers, Commit fails if there is no transaction pending).

Start

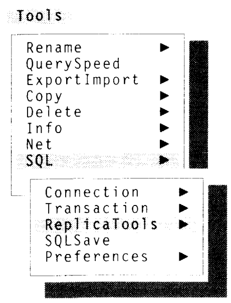


Tools | SQL | Transaction | Start lets you begin a transaction on a database server.

Use Start if your database server requires that you explicitly start a new transaction when you begin your session or after you conclude the current transaction with Commit or RollBack. If your server does this automatically, Start has no effect.

If your server does not automatically begin transactions, you must use Start if you want to be able to roll back your changes. See your server-specific addendum for more information.

ReplicaTools



Tools | SQL | ReplicaTools lets you perform basic file operations (rename, copy, and delete) on replicas of remote tables.

When you use the standard Tools | Copy and Tools | Delete commands, Paradox assumes they're to be performed on remote tables and their replicas. With ReplicaTools, you can perform operations on a replica without affecting the remote table. ReplicaTools do not affect local Paradox tables.

For example, when you delete a remote table (using Tools | Delete), you delete the remote table and its replica. When you delete a replica using Tools | SQL | ReplicaTools | Delete, however, you delete only the replica, not its remote table.

Use Tools | ReplicaTools | Rename to rename a replica. (You cannot rename a replica with Tools | Rename.)

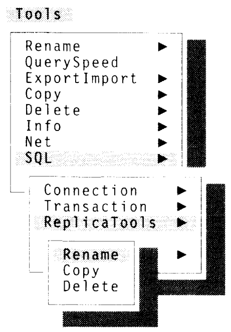
If the replica is encrypted, you will need to supply the proper password before you rename, copy, or delete it.

The ReplicaTools menu offers the following options:

Rename	Renames a replica.
Copy	Duplicates a replica.
Delete	Erases a replica you no longer need.

You would use the Rename option from the ReplicaTools menu, for example, to give a more appropriate name to a replica. You would use Copy to give another user access to a remote table by copying the replica to their working directory or to a shared directory. You would use Delete to delete a replica you no longer need.

Rename



Tools | SQL | ReplicaTools | Rename lets you assign a new name to a replica. This command (like the other ReplicaTools commands) renames the replica but not its remote table.

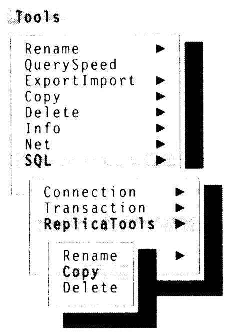
To rename a replica,

1. Choose Tools | SQL | ReplicaTools | Rename from the Paradox Main menu.
2. Type the name of the replica you want to rename, or choose it from the list box.

   Type the new name for the replica, then press *Enter*. If you type the name of an existing replica as the new name, Paradox displays an error message, and you'll have to specify a different name. If you type the name of an existing local table, Paradox asks if you want to cancel the operation or replace the local table.

   When you're through, Paradox renames the replica.

Copy



Tools | SQL | ReplicaTools | Copy lets you make a copy of a replica without affecting the remote table. You can copy the replica to a different directory or to a different name in the current directory. For example, you can give another user access to the remote table by copying its replica into a shared directory. The other user will still need read privileges to access the remote table with their user name and password.

To copy a replica,

1. Choose Tools | SQL | ReplicaTools | Copy from the Paradox Main menu.

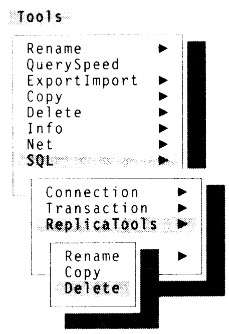
2. Type the name of the replica you want to copy or choose it from the list box.

3. Type the name (and the complete path name, if you want to put the copy in a different directory) for the target replica. If you type the name of a table that already exists in that location, Paradox gives you an error message, and you will have to specify a different replica name.

   Paradox copies the replica and all of its associated local objects, such as forms and reports.

Tools | SQL | ReplicaTools | Copy does not copy the associated remote table; it only affects Paradox access to it. If you want to copy the remote table as well, use the Tools | Copy command, discussed earlier in this chapter.

Delete



Tools | SQL | ReplicaTools | Delete lets you delete a replica without affecting its associated remote table. You can delete a replica if you no longer want to use it to access its remote table.

To delete a replica,

1. Choose Tools | SQL | ReplicaTools | Delete from the Paradox Main menu.

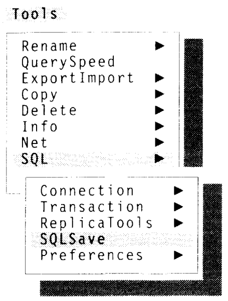
2. Type the name of the replica you want to delete, or choose it from the list box.

3. To protect against accidental deletion, Paradox asks you to confirm whether you really want to delete this replica. Choose Cancel to abandon the delete operation, or OK to proceed.

   Paradox deletes the replica and all of its associated local objects (such as forms and reports).

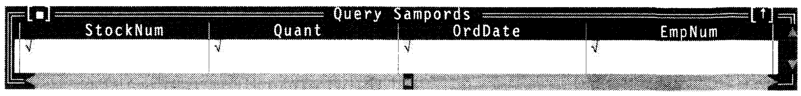
This operation does not delete the remote table, but it eliminates access to the remote table through this particular replica (you can create another replica with the SQL Setup Program). If you want to delete the remote table as well, use the Tools | Delete command, discussed earlier in this chapter.

SQLSave

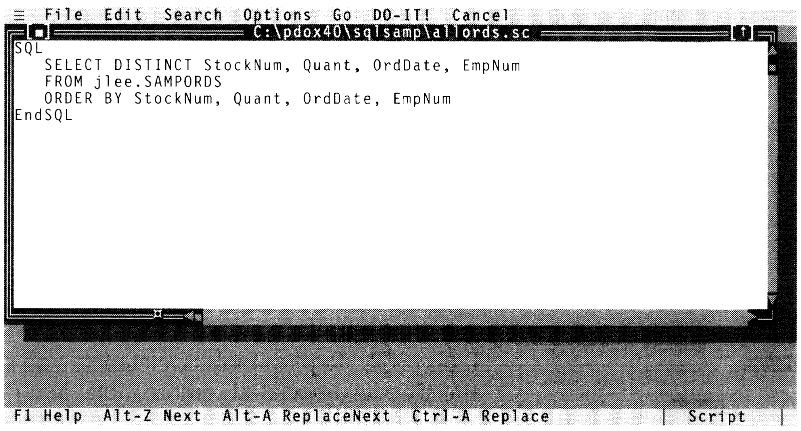


Tools | SQL | SQLSave lets you save the SQL translation of a query to a PAL script. The resulting script contains a SQL statement enclosed in a PAL SQL...ENDSQL command.

You can use SQLSave to save a query (created by choosing Ask from the Paradox Main menu) for later use. For example,



If you saved this query using SQLSave, you'd get a PAL script that looks like this:


≡ File Edit Search Options Go DO-IT! Cancel
C:\pdox40\sqlsamp\allords.sc
F1 Help Alt-Z Next Alt-A ReplaceNext Ctrl-A Replace | Script |

```
SQL
   SELECT DISTINCT StockNum, Quant, OrdDate, EmpNum
   FROM jlee.SAMPORDS
   ORDER BY StockNum, Quant, OrdDate, EmpNum
EndSQL
```

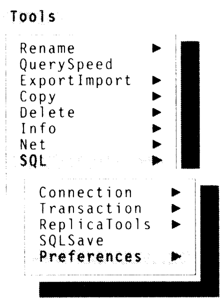
SQLSave lets you use *query-by-example* (QBE) to automatically construct SQL queries and include them in your PAL applications. You can also run saved scripts using Scripts | Play.

SQLSave is similar to the Scripts | QuerySave menu command. SQLSave saves a query as a SQL statement while QuerySave saves it as a PAL query image. If you save a query as a PAL query image, it is converted to SQL each time you run the script. This can be useful if you intend to run the query against servers that use different SQL dialects.

If you save a query as a SQL statement, you can modify the script to use the special tilde (~) variable. For more information, see the SQL...ENDSQL command in Chapter 6.

SQLSave saves the SQL statement you see when you press *Alt-F2* ShowSQL on the *Query* form. For more information about *Alt-F2* ShowSQL, see Ask, earlier in this chapter.

Preferences

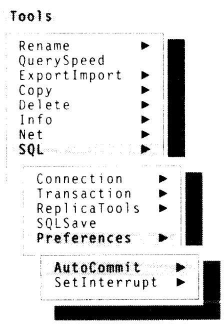


Tools | SQL | Preferences lets you modify global SQL settings that determine how other Paradox features work.

When you choose Preferences from the SQL menu, you see the following options:

AutoCommit	Determines whether Paradox saves changes to remote tables automatically. AutoCommit also affects locking after queries (see your server-specific addendum).
SetInterrupt	Determines whether pressing *Ctrl-Break* interrupts the current SQL operation in PAL scripts.

AutoCommit



Tools | SQL | Preferences | AutoCommit lets you decide whether Paradox automatically saves your changes to remote tables. By default, AutoCommit is set to Yes.

In the native Paradox environment, Paradox automatically saves your changes as you work. In the SQL environment, Paradox automatically commits certain kinds of changes (for example, when you create or delete a table on the database server). Paradox can also automatically save other remote table operations, or you can tell Paradox to accumulate changes until you're ready to explicitly commit (save) or roll back (undo) the transaction.

AutoCommit affects remote operations only. AutoCommit does not affect any *implicit commits* issued by the database server. AutoCommit does not affect PAL scripts that rely on embedded SQL programs (using SQL...ENDSQL) to perform remote operations. You must always issue an explicit SQLCOMMIT to save or SQLROLLBACK to undo your changes from within a PAL script.

When you choose AutoCommit from the Preferences menu, you see two options, Yes or No.

AutoCommit = Yes

If you set AutoCommit to Yes (the default setting), Paradox automatically commits any changes made to the remote table. You will not need to save your changes manually. You will not, however, be able to roll back any of the changes made to remote tables.

Even when AutoCommit is set to No, Paradox automatically issues a commit before and after the following SQL operations:

- CREATE
- COPY
- DELETE

AutoCommit = No

If you set AutoCommit to No, Paradox will not save any changes to remote tables until it receives a COMMIT from

- a menu command (see Tools | SQL | Transaction | Commit).
- a command in a PAL script (see SQLCOMMIT in Chapter 6).
- a SQL statement that has an explicit COMMIT. Even when AutoCommit is set to No, Paradox automatically issues a commit before and after the following SQL operations:
  - CREATE
  - COPY
  - DELETE

When AutoCommit=No, you can roll back changes.

With AutoCommit set to No, you can roll back (abandon) any changes made to remote tables (if your server has not committed your changes automatically). Changes can be rolled back in one of five ways:

- from the menu (see Tools | SQL | Transaction | RollBack)
- from within a PAL script (see SQLROLLBACK in Chapter 6)
- after an Empty command
- from an implicit rollback issued by the server
- if you break the server connection

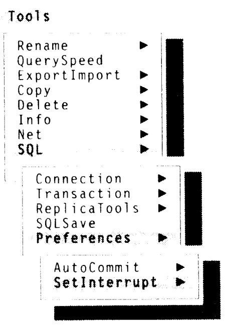
You must use Tools | SQL | Transaction | Start to start a transaction after setting AutoCommit to No, if required by the database server.

Important Paradox does not commit the changes on the current server, and the server will roll back the current transaction, if you break the connection in any of the following ways:

- by selecting a different server connection
- by choosing Tools | SQL | Connection | Break
- by pressing *Ctrl-Break* during a remote operation
- by executing the PAL DOSBIG or RUN BIG commands
- by using *Alt-0* DOSBig
- by exiting Paradox

For more information, see the discussion in Tools | SQL | Transaction earlier in this chapter, the discussion of transaction processing and the SQLAUTOCOMMIT command in Chapter 6, and your server-specific addendum.

SetInterrupt



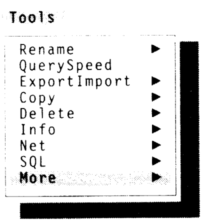
Tools | SQL | Preferences | SetInterrupt lets you control the effect of pressing *Ctrl-Break* during a PAL SQL application. By default, SetInterrupt is set to Yes. SetInterrupt affects only PAL scripts.

When you choose SetInterrupt from the Preferences menu during a PAL operation, you see two options:

Yes	Terminates the remote operation, rolls back the current transaction, breaks the current server connection, and returns control to your workstation.
No	*Ctrl-Break* is handled as in any Paradox operation—the PAL application breaks at the completion of the current PAL command.

For more information, see the SQLSETINTERRUPT command in Chapter 6.

More

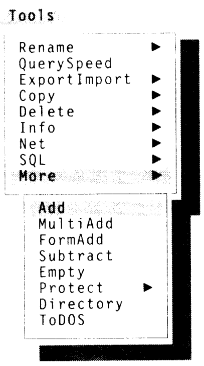


Tools | More lets you add data to remote tables, empty remote tables, and password protect replicas.

The following options on the More menu apply to remote tables:

Add	Adds records from one table to another, whether the tables are local or remote.
Empty	Deletes all records in a table, whether the tables are local or remote.
Protect	Password protects a replica or local table.

Add



Choose Tools | More | Add to add records from one table to another. Add lets you add

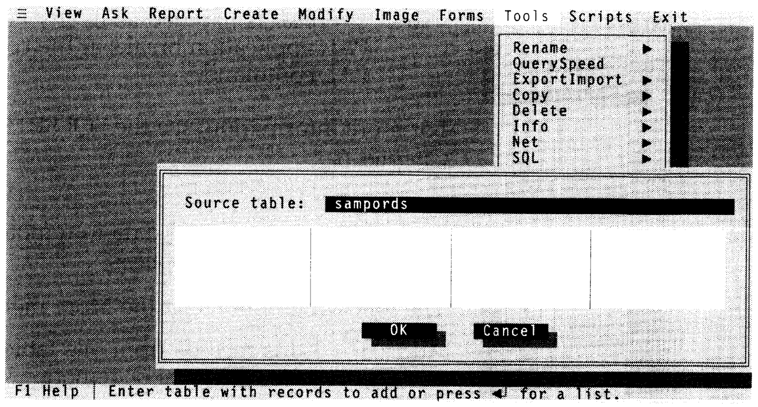
- ❏ a local table to a local table
- ❏ a local table to a remote table
- ❏ a remote table to a local table
- ❏ a remote table to a remote table, even from one server to another

For example, you could collect daily sales orders in a local Paradox table, then add them to a remote table at the end of the day. Or, you could collect shipping records on the database server, then add them to a local Paradox table at the end of the week. The tables must have compatible structures; that is, they must have identical column names, field types, and field order.

Also, the Update option lets you update records in indexed remote tables with matching records from a local table. Paradox overwrites the (non-key) field values in a record in the target (remote) table with the values of a record from the source (local) table if they have matching key values.

To add all the records from one table to another,

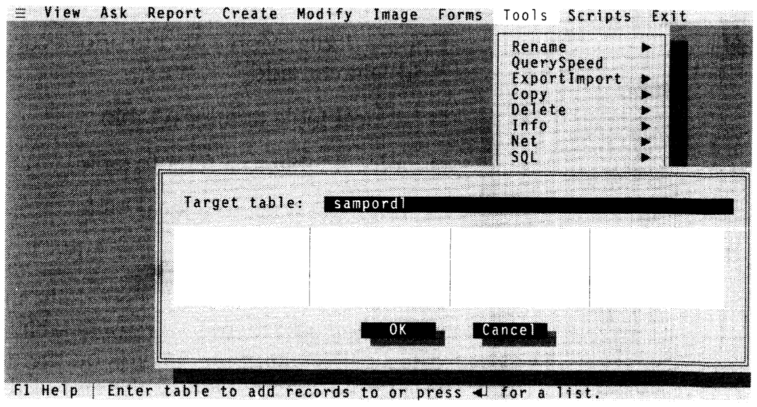
1. Choose Tools | More | Add from the Paradox Main menu.
2. Type the name of the source table (the one containing the records you want to add to the target table), or choose it from the list box. Remember that the source and target tables must have compatible structures (that is, identical field names, field types, and field order).



Source table: sampords

F1 Help | Enter table with records to add or press ◄┘ for a list.

3. Type the name of the target table to which you want to add the records, or choose it from the list box.



Target table: sampordl

F1 Help | Enter table to add records to or press ◄┘ for a list.

4. If the target is an indexed table, Paradox asks you whether you want to add the records from the source table as new records in the target table (NewEntries) or to overwrite records in the target table that have the same key (Update).

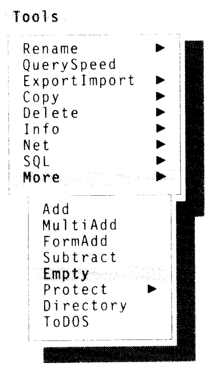
Paradox adds the records from the source table to the target table and creates a *Changed* table.

- As in native Paradox, the Add command is an implicit update to the remote table if the latter is keyed.
- If the target table is a keyed table and you chose the NewEntries option in the previous step, Paradox collects the duplicate records in a local *Keyviol* table if key violations occur.
- If other problems occur (such as adding records with null values to a remote table with non-null fields, or some other data integrity conflict), Paradox generates a *Problems* table. For more about this, see the discussion of *Problems* tables in Chapter 17 of the Paradox *User's Guide.*
- If another user has a table lock on the remote table, you'll have to wait until the user holding that lock frees the data by committing or rolling back the transaction. Records are added when the lock is released.
- If the server connection fails or if you interrupt the process in any of the following ways:
  - by selecting a different server connection
  - by pressing *Ctrl-Break* during a remote operation
  - by executing the PAL DOSBIG or RUN BIG commands
  - by using *Alt-0* DOSBig
  - by exiting Paradox

  the records will not be added to the target table. If the target is a remote table, the server rolls back the changes.

For more information about Tools | More | Add, see Chapter 17 of the Paradox *User's Guide.*

Empty



Choose Tools | More | Empty to delete all the records in a table. Use Empty when you no longer need the data but want to retain the table structure for future use.

To empty a remote table,

1. Choose Tools | More | Empty.
2. Type the name of the table from which you want to delete all records or choose it from the list box.
3. To protect against accidental data loss, Paradox shows you the fully qualified name of the remote table, and asks you to confirm that you want to empty the table. Choose Cancel to stop or OK to continue. If you choose OK, Paradox deletes all the records in the remote table.

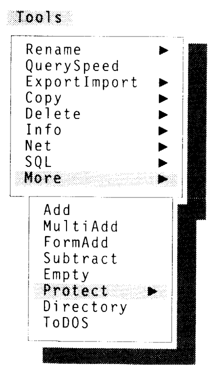
Table 5-3 illustrates the difference between Delete and Empty operations on remote tables.

Table 5-3 Differences between Delete and Empty operations

Action or condition	**Delete**	**Empty**
AutoCommit = Yes	Table deleted (dropped)	Table emptied (row deletion permanent)
AutoCommit = No	Table deleted (dropped)	Table emptied (row deletion is reversible)
Deletes table structure?	Yes	No
Can be rolled back before a COMMIT is executed?	No	Yes
Automatically commits whether AutoCommit is set to Yes or No?	Yes	No

For more information about Tools | More | Empty, see the discussion in Chapter 17 of the Paradox *User's Guide.*

Protect



Choosing Tools | More | Protect lets you password-protect a replica, controlling access to the remote table through Paradox.

The procedure for encrypting a replica is the same as the procedure for protecting a local Paradox table. For more information, see the discussion of Tools | More | Protect in Chapter 17 of the Paradox *User's Guide.*

Use the SQL statements GRANT and REVOKE to protect your remote data. See your database server manuals for more information.

CHAPTER 6

PAL commands and functions

Paradox SQL Link provides additional remote capabilities for existing PAL commands and introduces PAL commands and functions to help you manage your SQL session. Paradox SQL Link also enhances PAL by letting you include SQL statements in your PAL scripts using the SQL...ENDSQL command. Paradox sends your SQL statements directly to the database server, saves the results of any SELECT statement in a local *Answer* table, and lets you include PAL expressions and functions in your SQL statements.

Before you use these SQL extensions, you should be familiar with PAL programming. For a discussion of all aspects of PAL, see the *PAL Programmer's Guide*.

For a discussion of how to approach writing a PAL SQL Link application, see Chapter 8 of this manual.

Using SQL Link with PAL

You can use many of the PAL abbreviated menu commands with remote tables by typing the replica name of the remote table in place of a local Paradox table. The PAL commands in Table 6-1 accept replica names.

You can also use SQLISREPLICA() to see whether a table is local or a replica, then SQLCONNECTINFO() to see the location of the remote table, and SQLMAPINFO() to find the name of a remote table along with its column names and types.

For more information about these commands, see their respective entries later in this chapter.

Table 6-1 PAL commands used with Paradox SQL Link

Command	Description
ADD	Adds records from one table to another.
COPY	Copies a table.
CREATE	Creates a new table.
DELETE	Deletes a table.
EMPTY	Deletes all records in a table.
ERRORINFO	Provides comprehensive error information.
PROTECT	Password-protects a table.
QUERY...ENDQUERY	Places a query on the workspace.
REPORT	Prints a report for a table.

SQL Link PAL commands and functions

Table 6-2 lists the PAL commands that let you work with remote tables in the SQL environment.

Table 6-2 SQL Link PAL commands

Command	Description
ERRORINFO	Creates a dynamic array with information about the most recent error.
SHOWSQL	Simulates pressing *Alt-F2*.
SQL...ENDSQL	Sends passthrough SQL statements to the database server.
SQLAUTOCOMMIT	Specifies whether Paradox automatically commits every change on the server.
SQLBREAKCONNECT	Disconnects the current server connection.
SQLCLEARCONNECT	Clears the current server connection.
SQLCOMMIT	Commits changes made to remote tables on the database server.
SQLFETCH	Fetches a record from a pending query.
SQLMAKECONNECT	Connects to the server specified by the current connection.
SQLRELEASE	Releases the pending query.
SQLRESTORECONNECT	Restores the server connection saved with SQLSAVECONNECT.
SQLROLLBACK	Rolls back changes made to remote tables on the database server.
SQLSAVECONNECT	Temporarily saves the current server connection.
SQLSELECTCONNECT	Selects a server connection.

Command	Description
SQLSETINTERRUPT	Specifies whether pressing *Ctrl-Break* interrupts a remote operation.
SQLSTARTTRANS	Starts a transaction on the database server.

Table 6-3 lists the PAL functions that let you work with remote tables in the SQL environment. ISSQL() is the only SQL function that can be used when SQL Link is not active.

Table 6-3 SQL Link PAL functions

Function	Description
ISSQL()	Determines whether the user started Paradox with SQL on or off.
MENUPROMPT()	Returns Paradox's translation of the prompt issued by the database server.
SQLCONNECTINFO()	Returns connection information for a replica or for the current connection.
SQLERRORCODE()	Returns the most recent server-specific error code from the database server.
SQLERRORMESSAGE()	Returns the text of the most recent error message on the database server.
SQLISCONNECT()	Determines whether Paradox is currently connected to a server.
SQLISREPLICA()	Determines whether a table is a replica.
SQLMAPINFO()	Returns structural information for a remote table.
SQLVAL()	Returns valid SQL expression for use in SQL queries.

Each of the commands and functions is explained in detail later in this chapter.

Error handling

SQL Link adds error-handling features to Paradox that trap and report on remote errors on the database server. Remote errors trigger the Paradox error-handling routine, just as local errors do.

SQL Link adds remote error codes and messages to the PAL ERRORCODE() and ERRORMESSAGE() functions. Paradox intercepts errors from the database server and assigns its own standard codes and messages to them. Many server error codes can be mapped to a single Paradox error code; refer to Table 6-6 for a listing of Paradox's error codes and their explanations.

SQL Link also adds two functions, SQLERRORCODE() and SQLERRORMESSAGE(), to trap server errors directly.

SQLERRORCODE() returns the error code generated by the database server as a character string. Codes differ between server products.

SQLERRORMESSAGE() returns the actual error message generated by the database server. Refer to your database server documentation for more information about the codes and messages for your specific server. Refer to Table 6-6 for a listing of Paradox's error codes and their explanations.

SQLERRORCODE() and SQLERRORMESSAGE() retrieve information about remote errors only; use ERRORCODE() and ERRORMESSAGE() to trap errors that occur in the local Paradox environment.

You can add these features to your *Errorproc*. For example, the following error-handling procedure displays the server-specific error code and quits the application whenever an error occurs:

```
PROC AlwaysQuitErrorProc()
   IF (ERRORCODE() >= 1000)
      THEN QUIT SQLERRORCODE() + ":" + SQLERRORMESSAGE()
                               ; show the SQL message
      ELSE QUIT ERRORMESSAGE()     ; any other error
   ENDIF
ENDPROC
```

In this error-handling procedure, if the error is a SQL error, a flag is set to notify the application. If it's any other error, the procedure quits the application.

```
PROC FlagErrorProc()
   IF (ERRORCODE() < 1000)
      THEN QUIT ERRORMESSAGE()     ; any non-SQL error is fatal
   ENDIF
   SQLErrorFlag = True             ; flag error to application
   RETURN 1                        ; just continue
ENDPROC
```

For more information about error handling in PAL, see the "Error procedures" section in Chapter 7 of the *PAL Programmer's Guide*. For more information about Paradox SQL Link error-handling functions, see the descriptions of ERRORCODE(), ERRORMESSAGE(), and SQLERRORMESSAGE() later in this chapter.

Transaction processing

Paradox SQL Link lets you use the principles of transaction processing. In the SQL environment, an operation or series of operations (or changes) is called a transaction and is treated as a single unit of work. For example, a transaction can consist of a single operation, such as adding records from one table (*Table1*) to another (*Table2*) where the destination table (*Table2*) resides on the database server:

```
ADD "Table1" "Table2"    ; single-operation transaction
```

A transaction can also consist of a sequence of related operations on the database server, such as the following:

```
ADD "Table3" "Table2"
EMPTY "Table3"
```

Some servers require that you start a new transaction explicitly when you begin your remote session or after you've completed a transaction, while others do this for you automatically (see your server-specific addendum for more information). Paradox SQL Link provides a command for PAL applications, SQLSTARTTRANS, that begins a new transaction on the server, if your server has not already started a transaction for you.

At the conclusion of a transaction, you either save your work (*commit*), or abandon your changes (*rollback*). You might want to roll back your changes if, for example, an error occurred during the transaction that compromised the integrity of your data. SQL Link adds two commands, SQLCOMMIT and SQLROLLBACK, so you can save or abandon changes to remote tables.

Paradox automatically commits remote changes after each operation if SQLAUTOCOMMIT is set to Yes (the default). If you set SQLAUTOCOMMIT to No, Paradox lets you commit or roll back your changes explicitly. Paradox commits certain operations automatically, regardless of the SQLAUTOCOMMIT setting. These operations—the data-definition commands CREATE, COPY, and DELETE—cannot be rolled back. Paradox never commits changes resulting from SQL...ENDSQL commands automatically; you must always issue an explicit COMMIT or ROLLBACK.

The following example shows one way that transaction processing works. In this example, Paradox quits the application if any of the changes fails, and commits only after the last successful transaction. The first step is to define an error procedure that always tries to roll back the pending transaction, then quits the application.

```
PROC AlwaysRollbackAndQuit()
   PRIVATE Msg
   IF ISASSIGNED(InErrProc)         ; first make sure you don't call
      THEN RETURN 1                 ; this error proc from within
                                    ; the error procedure!
      ELSE InErrProc = True         ; make sure if next command fails,
   ENDIF                            ; ErrorProc is ignored.
   IF (ERRORCODE() >= 1000)
      THEN Msg = SQLERRORMESSAGE()  ; get the SQL message
         SQLROLLBACK
      ELSE Msg = ERRORMESSAGE()     ; any other error
   ENDIF
   RELEASE VARS InErrProc
   QUIT Msg
ENDPROC
```

In the next step, the following procedure performs two operations on the database. Assume that all the tables are remote. The two operations will succeed or fail together. If a SQL error occurs, the error procedure will quit.

```
PROC DoTransactionWithQuit()
   ErrorProc = "AlwaysRollbackAndQuit" ErrorProc()  ;assign error procedure
   SQLSTARTTRANS                  ; server might require
   IF Retval                      ; explicit transaction start
      THEN
         SQLAUTOCOMMIT No         ; turn Paradox commits off
         ADD "WeekOrd" "Orders"
         EMPTY "WeekOrd"
         SQLCOMMIT                ; all operations succeeded,
                                  ; commit this transaction
         SQLAUTOCOMMIT Yes        ; turn autocommit on again
   ENDIF
   RETURN Retval                  ; return Retval for evaluation
ENDPROC
```

As previously mentioned, the following commands commit pending transactions automatically, regardless of the setting of AutoCommit:

- CREATE REMOTE
- COPY REMOTE
- DELETE (on a remote table)

For more information about PAL commands relating to transaction processing see SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS later in this chapter.

Organization and notation

Each PAL SQL command or function is presented separately, in alphabetical order. The first remark in each entry explains what each command or function does. Each command or function contains the following sections:

Syntax	Describes the format of the command and any required or optional arguments, as outlined in the following section, "Notation."
Description	Describes the command, what it does, and its arguments.
Use	Discusses techniques for using the command, and notes any special cases you should know about.
Example	Contains programming examples (with comments) using the command or function.
See also	Refers you to related commands, functions, and applicable sections of the Paradox documentation.

Tables 6-4 and 6-5 list the parameters and keywords exclusive to Paradox SQL applications. See Table 1-2 in Chapter 1 of this manual for a concise description of the syntax conventions used in the command and function lookup sections. See also Appendixes B and C of the *PAL Reference* for parameters and keywords not listed here.

Notation

The Syntax subsection shows how each command and function is used. Syntax descriptions consist of several elements:

❐ *Keywords* (in uppercase letters) and required punctuation. Unless the optional or choice notation is used, you *must* type these exactly as shown, but any combination of uppercase and lowercase is acceptable.

❐ *Variable parameters* (in italics). You must replace each parameter with a valid expression, as described in the list of parameters in Table 6-4.

❐ *Optional elements* (enclosed in color brackets [ ]). For example,

CREATE [REMOTE]

shows that REMOTE is optional to the PAL command CREATE. Color brackets are used only for describing command syntax and are *not* to be typed.

❐ *Lists* (enclosed in color braces, with options separated by colored vertical bars). For example,

SQLSETINTERRUPT { Yes | No }

means to follow the SQLSETINTERRUPT command with one and *only one* of the choices shown—Yes or No.

The color elements (braces and vertical bars) are used only for describing command syntax; do *not* enter them. Do not confuse these color braces with regular braces ({ }) that are actually elements of PAL expressions, such as

```
Menu {Tools} {More} {Directory}
```

Table 6-4 PAL SQL variables

Parameter	**Description**
ColumnNumber	Any ordinal number representing a field in the remote table.
OptionName	A string specific to the connection, SQL dialect, remote table, or associated server product.
Expression	Any valid PAL expression. See the *PAL Programmer's Guide* for information on PAL expressions.
ParameterList	A comma-separated list of the connection parameters required for SQL connection.
ProductName	Code name of the server product.
ReplicaName	Any replica name.
SQLText	Any valid SQL statement.
TableName	The valid name of either a local table or the replica of a remote table.
Title	Connection name as shown on the SQL Connections screen.

Table 6-5 PAL SQL keywords

Keyword	**Description**
DESCRIPTION	Used in SQLCONNECTINFO()
DIALECT	Used in SQLCONNECTINFO()
NOFETCH	Used in SQL...ENDSQL
PRODUCT	Used in SQLSELECTCONNECT and SQLCONNECTINFO()
REMOTE	Used in COPY, CREATE
TITLE	Used in SQLSELECTCONNECT and SQLCONNECTINFO()
VALUES	Used in SQLSELECTCONNECT

ADD

Adds the records of one table to another.

Syntax

ADD *TableName1 TableName2*

TableName1 is the source table (either a local table or the replica of a remote table), and *TableName2* is the destination table (either a local table or the replica of a remote table).

Description

Like the equivalent Tools | More | Add menu command, the abbreviated PAL command ADD adds records in the source table to the destination table.

When SQLAUTOCOMMIT is set to Yes, Paradox commits the added records automatically. If an error occurs while you're adding the records, Paradox rolls back the transaction. When SQLAUTOCOMMIT is set to No, you'll need to explicitly commit the added records by using SQLCOMMIT, or if an error occurs, you'll need to explicitly roll back the transaction using SQLROLLBACK. If you use SQLBREAKCONNECT before SQLCOMMIT, the server automatically rolls back your changes.

Use

ADD lets you accumulate data in one table and transfer it to another when you're ready. For example, you could collect shipping data in a local Paradox table and, at the end of the week, add the data to a remote table on the database server.

If the destination table is indexed, ADD updates records with matching keys from the source table. ADD automatically selects the UPDATE option (see the discussion of Tools | More | Add in Chapter 5 of this manual). In this way, you can directly modify the data in existing records in a remote table. Paradox creates a local *Changed* table for those records that were changed on the server. If key violations occur, Paradox creates a local *Keyviol* table.

If other errors occur (any data integrity conflict), Paradox creates a *Problems* table. For more information about *Problems* tables, see Chapter 17 of the Paradox *User's Guide*.

To use ADD, Paradox must be in Main mode. Both tables must be specified and have compatible field structures (field name, order, and type), or a script error results.

Example

Suppose you accumulate daily sales records in a local Paradox table named *Tdysales* and, at the end of the day, you want to add this information to a remote table named *Allsales*. This example shows how to do it:

```
ADD "Tdysales" "Allsales"     ; adds records from local
                              ; table to remote table
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- Tools | More | Add in Chapter 5 of this manual
- ADD command in the *PAL Reference*

COPY

Makes a copy of a table and its family of objects.

Syntax

COPY *TableName1* [**REMOTE**] *TableName2*

TableName1 is the source table (either a local table or the replica of a remote table), and *TableName2* is the destination table (either local or remote).

Description

Like the equivalent Tools | Copy menu command, the abbreviated PAL command COPY copies a table to a new table and, if it has a primary index, copies the index too. If the source table has any other associated files (such as reports or forms), COPY duplicates these as well. You can copy a file from either a local or remote table, and you can copy to either a remote or local table.

If the destination for the copy is a server, COPY creates a replica. If *TableName2* is remote, you should make sure that the field names comply with the naming rules of your database server (do not use Paradox or server-reserved words). See your server-specific addendum for more information.

Note If you are creating a remote table, the source table structure must comply with the server's field-naming rules. This means that neither table can include embedded spaces or reserved words. In addition, the data-type boundaries (such as dates and numbers) must comply with the server's rules.

If you use the REMOTE keyword, you need to select a connection before you use COPY.

You cannot use an existing table name or replica name for the target table.

Even if SQLAUTOCOMMIT is set to No, if the target table is remote, Paradox issues a commit so that the COPY can occur. This means that you cannot roll back a COPY and cannot use COPY in the middle of a transaction. All work done before the COPY is committed at the start of the COPY. If necessary, start a new transaction after the COPY operation.

If a local Paradox table has secondary indexes, they will not be copied to the remote table. Although you can add secondary indexes to remote tables using passthrough SQL, the secondary indexes you create are not Paradox-compliant.

Copy is a single operation.

Paradox treats COPY as a single, atomic operation; that is, if the copy fails, the entire operation is aborted. For example, COPY fails if the source table contains data that does not match the type or range of columns in the target table, as would be the case if the source table contains a date that is valid in Paradox but not on the server database. If COPY fails, you can create a new target table, borrow the structure from the source table, and add the records from the source table to the target table. If an invalid record is detected, it is stored in a *Problems* table and the ADD proceeds.

Use

Use COPY to create a duplicate of a table. The duplicate is a snapshot of the original table at the time you copied it. The data in the duplicate table will not be automatically updated when that in the source table changes.

Example

Suppose you keep your master price list on a remote table (named *Pricelst*), and you want to use that information in a report involving several local Paradox tables. You could copy *Pricelst* to a local table:

```
COPY "Pricelst" "Prices"      ; Prices is local Paradox table
```

Suppose you collect daily sales information in a local Paradox table named *Tdysales* and, at the end of the day, want to copy that information to a remote table named *Daysales* that users access on the database server. Here's how you do it:

```
COPY "Tdysales" REMOTE "Daysales"
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- Tools | Copy in Chapter 5 of this manual
- COPY command in the *PAL Reference*

CREATE

Creates a new table.

Syntax

1. **CREATE [REMOTE]** *TableName FieldNameList*

   to create a new table with the fields specified in the list

2. **CREATE [REMOTE]** *TableName2* **LIKE** *TableName1*

   to create *TableName2* with the same structure as *TableName1*

Description

Like the equivalent Create menu command, the abbreviated PAL command CREATE creates a new table, either local or remote. If the new table is remote, Paradox creates its replica so it can find the remote table in the future. To protect you from accidentally overwriting an existing table, Paradox won't let you specify the name of a remote table that already exists.

Using syntax form 1, you can create a table by specifying a list of fields. You can define up to 255 fields. If you specify key fields with an asterisk (*), Paradox creates a unique index for the table. You can also create a table (borrowing the structure of an existing table) by using syntax form 2.

If you use the REMOTE keyword, you need to select a connection before you use CREATE. You should make sure that the field names in the new remote table comply with the naming rules of your database server (do not use Paradox or server-reserved words). See your server-specific addendum for more information.

Note Spaces are not allowed in fields of remote tables. If you use CREATE REMOTE...LIKE... using a local table containing fields with embedded spaces, this operation fails.

Even if SQLAUTOCOMMIT is set to No, if the table you're creating is remote, Paradox issues a COMMIT before and after the operation so that the CREATE can occur. This means that you cannot roll back a CREATE and cannot use CREATE in the middle of a transaction. All work done before the CREATE is committed at the start of the CREATE. If necessary, start a new transaction after the CREATE operation.

To let other users access the new remote table, use Tools | SQL | ReplicaTools | Copy to copy the replica to a shared directory. Then use SQL commands to grant users access to the new remote table.

Use

Use CREATE to define a new table. Suppose, for example, you want to distribute an application that creates remote tables during installation. You can use CREATE to generate the remote tables using the structure of local Paradox tables:

```
CREATE REMOTE "remtabl1" LIKE "loctabl1"
CREATE REMOTE "remtabl2" LIKE "loctabl2"
CREATE REMOTE "remtabl3" LIKE "loctabl3"
```

Example

This example creates a remote table called *Invoices* with a specified list of fields, including an index on the key field InvoiceNbr (an asterisk denotes the key field):

```
CREATE REMOTE "Invoices"
"InvoiceNbr"   : "S*",
"CustomerNbr"  : "S",
"InvoiceDate"  : "D",
"TotalCharges" : "$",
"Comments"     : "A20"
```

The following example creates a remote table called *Febsales* using the structure of another remote table called *Jansales*:

```
CREATE REMOTE "Febsales" LIKE "Jansales"
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- Create in Chapter 5 of this manual
- CREATE command in the *PAL Reference*

DELETE

Removes a table and its associated files.

Syntax

DELETE *TableName*

TableName is the name of the table (either a local table or the replica of a remote table) you want to delete.

Description

Like the equivalent Tools | Delete menu command, the abbreviated PAL command DELETE removes a table and all associated files. You can delete either a local or remote table. Unlike the menu command, DELETE does *not* ask the user for confirmation. If *Tablename* is a replica, the remote table, the replica, and all family members will be deleted (the remote table will be dropped).

Use

You delete a table to free up disk space when you no longer need it (such as a temporary table). Since DELETE does not ask the user for confirmation, your program should do so explicitly. See the example for DELETE in the *PAL Reference*.

When you delete a remote table, Paradox deletes the replica and all corresponding local objects as well.

If the target table you're deleting is remote, Paradox issues a COMMIT before and after the DELETE operation (even if SQLAUTOCOMMIT is set to No). This means that you cannot roll back a DELETE and cannot use DELETE in the middle of a transaction. All work done before the DELETE is committed at the start of the DELETE. If necessary, start a new transaction after the DELETE operation.

Example

This example shows how to delete a remote table called *Checks*:

```
DELETE "Checks"
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- Tools | Delete in Chapter 5 of this manual
- DELETE command in the *PAL Reference*
- Table 5-3 in Chapter 5 of this manual, which shows the difference between EMPTY and DELETE operations on remote tables

EMPTY

Removes all records from a table.

Syntax

EMPTY *TableName*

TableName is the name of the table (either a local table or the replica of a remote table) you want to empty.

Description

Like the equivalent Tools | More | Empty menu command, the abbreviated PAL command EMPTY removes all records from the specified table. You can EMPTY records in either a local or remote table. (When emptying a remote table, the SQL command DELETE FROM *TableName* is sent to the server.) Unlike the menu command, EMPTY does *not* ask the user for confirmation.

Use

Use EMPTY to delete all the information in a table without deleting the table itself. You can use EMPTY to clear data from a temporary table used for reporting or editing.

When SQLAUTOCOMMIT is set to Yes, Paradox automatically commits the removal of records. If an error occurs while you're deleting records, Paradox rolls back the transaction. When

SQLAUTOCOMMIT is set to No, you'll need to explicitly commit the deletions with SQLCOMMIT. If an error occurs, you'll need to explicitly roll back the transaction using SQLROLLBACK.

Example

Suppose you print a sales report on a weekly basis. To gather data for your weekly report, you combine sales data from several local and remote tables and store it in a remote table named *Salesrpt*. After you have generated and printed the report, you can empty the *Salesrpt* table as shown in the following example:

```
MESSAGE "Empty the sales report table (Y/N)? "
ACCEPT "A1" TO Response
IF ((Response = "Y") OR (Response = "y"))
   THEN EMPTY "Salesrpt"   ; deletes all records
ENDIF
```

This example empties the contents of a remote table called *Invntory*:

```
EMPTY "Invntory"         ; deletes all records
```

See also

- ❒ SQLAUTOCOMMIT, SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- ❒ Tools | More | Empty in Chapter 5 of this manual
- ❒ EMPTY command in the *PAL Reference*
- ❒ Table 5-3 in Chapter 5 of this manual, which shows the difference between EMPTY and DELETE operations on remote tables

ERRORCODE ()

Returns the Paradox code of the most recent error resulting from a local or remote operation.

Syntax

ERRORCODE ()

ERRORCODE() takes no arguments. It returns a numeric value indicating the category of the most recent run-time error or error condition in the local Paradox environment or on the database server. If no error has occurred since the beginning of the Paradox session, ERRORCODE() returns 0 (zero).

Description

ERRORCODE() lets you see what kind of error has occurred. It differs from most other functions in that it is dynamic; its value is context-specific. In conjunction with the global variable *Errorproc*, ERRORCODE() lets you build an error-handling routine into your application. Refer to Table 6-6 for a list of error codes and messages. All SQL-specific error codes are in the range starting at 1000.

ERRORCODE() reports on both local and remote errors. A single Paradox error code can correspond to many related server error codes; refer to the results from the SQLERRORCODE() and SQLERRORMESSAGE() functions and your server manuals for more specific information. Use SQLERRORCODE() to trap server-specific errors that occur in the remote SQL environment. Check SQLERRORMESSAGE() for the server-specific error message from the server.

Table 6-6 Paradox error codes for the SQL environment

ERRORCODE()	ERRORMESSAGE()
1000	General SQL error—check SQLERRORMESSAGE() to return the server error message
1001	Network error
1002	Deadlock on server
1003	User aborted (*Ctrl-Break*)
1004	Not enough memory to complete operation
1005	Communication error
1006	Connection failed
1007	Insufficient access privileges or incompatible locks
1008	Object already exists
1009	Object name invalid
1010	General create error
1011	Database or disk full
1012	Object does not exist
1013	Column type or usage invalid
1014	Remote key violations (SQL...ENDSQL only)
1015	Syntax error (SQL...ENDSQL only)
1016	Copy failed
1017	Number of authorized users exceeded
1018	Replica inconsistent with remote table
1019	A replica named *ReplicaName* already exists

Use

Use ERRORCODE() to create an error-handling routine for remote operations within your application. For examples and more information, see the discussion in "Error handling" earlier in this chapter.

See also

- ERRORMESSAGE(), SQLERRORCODE(), and SQLERRORMESSAGE()
- the discussion of error handling earlier in this chapter and in Chapter 7 of the *PAL Programmer's Guide*

- ERRORCODE() in the *PAL Reference*
- your server-specific addendum

ERRORINFO

Creates a dynamic array with information about the latest script error.

Syntax

ERRORINFO TO *DynArrayName*

Description

ERRORINFO creates a dynamic array with information about the latest run-time error. This dynamic array contains indexes for error attributes and their values.

Table 6-7 ERRORINFO array elements

Index	Definition
SCRIPT	The name of the script that caused the error
LINE	The line number in the script where the error occurred
POSITION	The character position in the script where the error occurred
CODE	Category code of most recent error; same as the value returned by ERRORCODE()
USER	Name of the user who has locked an object; same as the value returned by ERRORUSER()
MESSAGE	Text of the most recent error message; same as the value returned by ERRORMESSAGE()
PROC	The name of the current PROC; blank if no procedure
SQLERRORMESSAGE	Text of the most recent SQL error message if SQL is loaded (same as the value returned by SQLERRORMESSAGE()); blank otherwise
SQLERRORCODE	Number that represents the code of the most recent SQL error if SQL is loaded (same as the value returned by SQLERRORCODE()); 0 (zero) otherwise

Use

ERRORINFO returns only information about run-time errors. Script errors and syntax errors won't update any of the information returned by ERRORINFO.

See also

- DYNARRAY command
- ERRORCODE(), ERRORMESSAGE(), and ERRORUSER() functions

- Chapter 7 of the *PAL Programmer's Guide* for information about error handling

ERRORMESSAGE ()

Returns the Paradox message for the most recent error resulting from a local or remote operation.

Syntax

ERRORMESSAGE ()

ERRORMESSAGE() takes no arguments. It returns a string containing the text of the most recent run-time error or error condition in the local Paradox environment or on the database server. This is the same message that Paradox displays on the message line after the error occurs. If no error has occurred since the beginning of the Paradox session, ERRORMESSAGE() returns a blank (" ") string.

Description

ERRORMESSAGE() is used to retrieve the message generated by Paradox when the last error occurred. ERRORMESSAGE() reports on local and remote errors. Use SQLERRORMESSAGE() to return the server-specific message. See Table 6-6 for a list of error codes and messages.

Usage

Use ERRORMESSAGE() to create an error-handling routine for remote operations within your application. See the discussion in "Error handling" earlier in this chapter.

See Also

- ERRORCODE(), SQLERRORCODE(), and SQLERRORMESSAGE()
- the discussion of error handling earlier in this chapter and in Chapter 7 of the *PAL Programmer's Guide*
- ERRORMESSAGE() function in the *PAL Reference*
- your server-specific addendum

ISSQL ()

Determines whether SQL Link is running.

Syntax

ISSQL ()

Description

ISSQL() lets you determine whether SQL Link is installed and enabled. SQL capabilities are enabled by default once you install SQL

Link. ISSQL() returns False if SQL Link is not installed or if you start Paradox from the DOS prompt with:

paradox -sql off

This is the only SQL Link function that you can use when SQL Link is not enabled.

Use

ISSQL() lets you determine whether the user started Paradox with SQL Link installed and enabled, allowing you to run remote operations in your application. You should check this once at the beginning of any PAL script or application in which a remote operation occurs (and any script that calls another script that runs a remote operation).

Example

This example uses ISSQL() at the beginning of a PAL script to verify that SQL Link is enabled before calling a procedure that uses a remote operation:

```
IF ISSQL()                  ; if SQL enabled
   THEN DoRemote()          ; can run remote operations
   ELSE
      MESSAGE "Cannot continue - SQL Link is not activated."
   RETURN
ENDIF
```

See also

❐ "Starting Paradox" in your server-specific addendum

MENUPROMPT ()

Returns the text of the current type-in prompt.

Syntax

MENUPROMPT ()

MENUPROMPT() takes no arguments. It returns a string value containing the text of the current Paradox type-in prompt, or the string "Error" if a dialog box with a type-in prompt is not being displayed.

Description

If a Paradox menu dialog box is currently displayed, MENUPROMPT() returns a string containing the type-in label. For example, MENUPROMPT() returns the prompt that appears in a dialog box when the user accesses a replica and is not connected to the server associated with that replica.

This function applies only to Paradox menu dialog boxes and not to dialog boxes created with SHOWDIALOG. MENUPROMPT() is only available for dialog boxes that request a type-in response; it does not apply to menus or list boxes. If MENUPROMPT() is used in an inappropriate context, it returns the string "Error".

Example

```
{ASK} SELECT "ORDERS"           ; query remote table
IF MENUPROMPT()<>"Error"
   QUIT "Cannot run remote query. You" +
   "are not logged in to the server."
ENDIF
CHECK                           ;  place check marks in all fields
DO_IT!                          ;  execute remote query
```

See also

❒ MENUCHOICE() in the *PAL Reference*

QUERY

Places a query statement on the workspace.

Syntax

QUERY
   QueryImage
ENDQUERY

QueryImage represents the query statement.

Description

This command places a query statement on the workspace but does not execute it. (Follow the ENDQUERY keyword with DO_IT! to execute the query.) Note that blank lines must separate *QueryImage* from the QUERY and ENDQUERY keywords.

The default sort order for the fields in the *Answer* table is set in the Custom Configuration Program (CCP) and can be changed with the SETQUERYORDER command. You can use QUERYORDER() to check the current default setting.

To use the QUERY command, compose the query interactively in Paradox, then use Scripts | QuerySave to save it as a script. The saved *Query* image will be properly enclosed by the QUERY and ENDQUERY keywords. Then, either play the query script from your script or use the Editor to insert it into your script.

QueryImage is straight ASCII text, which can be edited. It is preferable to make all but minor edits to a saved query; use Scripts | Play to redisplay the query, edit the query on the workspace, then use Scripts | Query Save to save it again. Paradox validates changes you make to a query with the latter method; it is easy to make an error if you edit the ASCII query statement in your script.

In the query script, checked fields are indicated by the keywords CHECK, CHECKPLUS, CHECKDESCENDING, or GROUPBY. Example elements are preceded by an underscore (_).

Use

You can use PAL variables in a query by preceding the variable name with a tilde (~). Because PAL uses an underscore character to represent an example element, do not use an underscore in a variable name that will be used in a query.

Example

This script looks up the customer record for the customer whose name is stored in the variable *name*:

```
name = "Jones"
QUERY                       ; put the query on the workspace

 Customer  | Cust ID  | Last Name   | Init  | Street | City  |
           | Check    | Check ~name | Check | Check  | Check |

ENDQUERY
DO_IT!                      ; execute the query
IF (ISEMPTY("Answer"))      ; none found
   THEN MESSAGE name + " was not found."
   ELSE
      FORMKEY               ; go into form view
      WAIT RECORD           ; let user examine record
         MESSAGE "Press F2 when done."
      UNTIL "F2"
ENDIF
```

See also

- SETRESTARTCOUNT command
- QUERYORDER() function
- Chapter 9 of the *PAL Programmer's Guide* for information about creating query scripts
- Chapters 5 and 6 of the Paradox *User's Guide* for information about queries

REPORT

Prints a report.

Syntax

REPORT *TableName ReportName*

TableName is the name of the table (either a local table or the replica of a remote table) on which you want to report, and *ReportName* is the name of the report you want to use. Use "R" for a standard report, or "1" through "14" for a custom report.

Description

Like the equivalent Report | Output | Printer menu command, the abbreviated PAL command REPORT sends a specified report for *TableName* to the printer. You can create custom reports for a remote table using the Report | Design command.

Use

REPORT lets you print the contents of a remote table. You can report on any single remote table, but you cannot use REPORT on multiple remote tables simultaneously. You can, however, query a remote table and run a multi-table report using the *Answer* table with other local tables.

Depending on the size and complexity of the remote table, it may take a long time to complete the specified report. If you are not familiar with the structure of the remote table that you want to report on, use the Tools | Info | Structure command on the replica to obtain this information before generating a report on the entire table. You can also use CALC COUNT ALL to find out how many records are in the remote table, or query the table to extract the records you are interested in, copy the report to the *Answer* table, and print the report.

Example

This example prints a standard report for a remote table called *Clients*:

```
REPORT "Clients" "R"
```

In the following example, REPORT prints a custom report, "2," created using Report | Design from the Paradox menu:

```
REPORT "Clients" "2"
```

See also

❒ REPORT command in the *PAL Reference*

SHOWSQL

Shows the SQL statement equivalent to the current Paradox query.

Syntax

SHOWSQL

Description

SHOWSQL returns the SQL code for the current replica query and displays it in a window. Performs the same function as pressing *Alt-F2*.

Use

SHOWSQL works only on replicas. It provides an easy way for you to view and debug QBE and SQL queries.

Example

The following example queries the *Accts* table and displays the equivalent SQL statement. The *Accts* table is a replica of a remote table.

```
ECHO NORMAL
QUERY
   Accts | Acct# | Name        |
         | CHECK | CHECK Smith |
ENDQUERY
Message "Viewing SQL; Press any key to continue"
SHOWSQL                          ; Show the user the SQL translation

GETEVENT KEY "ALL" TO the Event  ; Hold the SQL translation until a keypress
DO_IT!                           ;  Execute the remote query
```

See also

- "Viewing the SQL translation of your query" in Chapter 3

SQL...ENDSQL

Sends passthrough SQL statements to the database server.

Syntax

SQL [NOFETCH]
 { *SQLText* | *~Expression~* } ...
ENDSQL

Where *SQLText* is any valid SQL command and *Expression* is any valid PAL expression. Individually, *SQLText* and *Expression* can be valid SQL statements, or they can be combined to create a valid SQL statement.

Description

PAL sends one or more lines of text contained in SQL...ENDSQL directly to the server. PAL evaluates any valid PAL expression enclosed in tildes (~), converts the result to an alphanumeric string, then passes the string to the server.

You can also include PAL comments within the SQL statement, as in the following example:

```
SQL
   SELECT *      ; all fields
   FROM Orders   ; from Orders table
ENDSQL
```

Note Each SQL...ENDSQL command contains only one SQL statement; you cannot nest SQL statements.

The system variable *Retval* distinguishes between queries and other SQL statements. *Retval* is set to True if the SQL statement returns a result, and False if it doesn't. *Retval* is set to True for a SELECT statement even if the result is empty. If you do *not* use the NOFETCH keyword and the SQL statement returns a result, Paradox fetches the records and creates a local *Answer* table. If an *Answer* table is created, it's not displayed on the workspace. Use View from the Paradox Main menu to view the *Answer* table and see the resulting records.

If you use the NOFETCH keyword and the SQL statement returns a result, Paradox does *not* create an *Answer* table. Use SQLFETCH to manipulate these results and SQLRELEASE to release the query when you're done.

Before you use the SQL...ENDSQL statement, you must connect to a server. The SQL dialect you use must be compatible with this connection. If the SQL statement produces an answer, and the NOFETCH keyword is not used, then SQL...ENDSQL must be executed in Main mode; otherwise, the SQL NOFETCH...ENDSQL command can be used in any mode.

Use

You can decide when to save (SQLCOMMIT) or abandon (SQLROLLBACK) changes , regardless of the setting of SQLAUTOCOMMIT. Paradox never automatically commits remote changes resulting from SQL...ENDSQL commands. When you use SQL...ENDSQL, the database server checks the syntax of your SQL statements. Paradox then traps any run-time errors, which you can then retrieve in your error procedure using SQLERRORCODE() and SQLERRORMESSAGE().

Once you have connected to the server, you can send a SQL...ENDSQL command to the server from the PAL Menu. Press *Alt-F10* to display the menu, choose MiniScript, type in your SQL...ENDSQL command, and press *Enter*. Paradox sends the statement to the database server for processing and displays any error messages. If your command was a query (and you did not use NOFETCH), SQL Link retrieves any resulting data in an *Answer* table. You can view the *Answer* table to see the resulting records. For more information about the MiniScript command, see Chapter 8 of the *PAL Programmer's Guide.*

In a SQL statement, you can substitute any PAL expression by enclosing it in tildes (~). Paradox evaluates the expression first, then combines it with the rest of the statement and sends it as one string to the database server. For example,

```
Tbl = "Customer"
SQL
   SELECT *                  ; all fields
   FROM ~Tbl~                ; from Customer who
   WHERE totalordered > 1000  ; ordered more than $1000
ENDSQL
```

You can also use the UseSQL utility to send SQL statements to the server.

Example

You can enclose any valid SQL statement in the SQL...ENDSQL command. In this example, the command is on a single line:

```
SQL SELECT * FROM Orders WHERE State = 'CA' ENDSQL
```

As in the next two examples, the command can span several lines:

```
SQL
   ALTER TABLE Customer ADD (homephone CHAR(14))
ENDSQL
```

```
SQL
   SELECT *                          ; select all fields
   FROM Maillist                     ; from mail list for
   WHERE Age > 50                    ; those over 50 years old
   ORDER BY lastname, firstname, phone
ENDSQL
```

The following example demonstrates how to use the SQLFETCH command in conjunction with the SQL NOFETCH...ENDSQL and SQLRELEASE commands. We create an array of names (up to twenty names can be included) in the remote table *Names* for a given zip code, in last-name order. Each SQLFETCH fetches one record from the remote result. The field values are available to the PAL application in an array. The first element of the array is not used (assigned with a null " " string), and the rest of the array is filled with the column values, as in COPYTOARRAY. In this example, those values are *FirstName, Init,* and *LastName.*

```
MaxNames = 20
ARRAY FullName[MaxNames]
SQL NOFETCH                    ; query Names
   SELECT DISTINCT FirstName, Init, LastName
   FROM Names
   WHERE Zip = ~ZipCode()~     ; returns a Zip Code
   ORDER BY LastName, FirstName, Init
ENDSQL

FOR I FROM 1 TO MaxNames
  SQLFETCH Name
  IF RETVAL
    THEN FullName[I]=
      Name[2] + " " + Name[3] + " " + Name[4]
  ELSE
    QUITLOOP
  ENDIF
ENDFOR

SQLRELEASE                     ; release the pending query
```

In the next example, the user is prompted for a SQL command. The result is stored in a variable, then executed within SQL...ENDSQL:

```
@ 10,10
?? "Enter SQL command:"
ACCEPT "A50" TO SQLCommand
SQL
   ~SQLCommand~
ENDSQL
If Retval
   THEN VIEW "Answer"
ENDIF
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, SQLERRORCODE(), SQLERRORMESSAGE(), SQLFETCH, SQLRELEASE, SQLROLLBACK, SQLSELECTCONNECT, SQLSTARTTRANS, and SQLVAL()
- Tools | SQL | SQLSave in Chapter 5 of this manual
- Chapter 4 of this manual for information on UseSQL, the SQL command editor

SQLAUTOCOMMIT

Specifies whether Paradox automatically saves changes at the conclusion of every successful remote operation.

Syntax

SQLAUTOCOMMIT { Yes | No }

Description

Like the equivalent Tools | SQL | Preferences | AutoCommit menu command, setting SQLAUTOCOMMIT to Yes tells Paradox to commit changes at the conclusion of every successful remote operation. When you set it to No, the server accumulates changes until you are ready to commit them (with SQLCOMMIT) or roll them back (with SQLROLLBACK). SQLAUTOCOMMIT only affects changes on remote tables; Paradox automatically saves your changes to local tables.

SQLAUTOCOMMIT does not affect implicit commits issued by your server. Paradox also commits changes automatically after certain remote operations (such as COPY, CREATE, and DELETE), regardless of the SQLAUTOCOMMIT setting.

The SQLAUTOCOMMIT setting does not affect updates made using SQL...ENDSQL. You must always commit those changes explicitly. Also, setting SQLAUTOCOMMIT to No does not start a transaction. You might need to use SQLSTARTTRANS to do this if your server doesn't do it for you automatically.

Important If you set SQLAUTOCOMMIT to No, you must commit your changes on the database server before breaking a server connection, making a new server connection, or exiting Paradox. Otherwise, the server will roll back your changes when Paradox breaks the connection. The following operations break a connection:

- selecting a different server connection
- pressing *Ctrl-Break* during a remote operation
- executing the PAL DOSBIG or RUN BIG commands

- using *Alt-0* DOSBig
- exiting Paradox

For an introduction to transaction processing concepts, see the discussion in Chapter 2 and the "Transaction processing" section earlier in this chapter.

Use

Set SQLAUTOCOMMIT to Yes if you want Paradox to commit changes for you automatically, or to No if you want to commit or roll back your changes once you're sure all operations in the transaction were successful.

The SQLAUTOCOMMIT settings also affect locking after queries. See your server-specific addendum for details.

Example

Suppose you want to add your daily sales data, collected in a local Paradox table named *Dlysales*, to two remote tables: *Mthsales* containing monthly sales data, and *Ytdsales* containing all sales to date. Once added, you then want to delete the daily records to clear the table for tomorrow's activity.

This example shows how Paradox automatically commits changes to remote tables after each operation when SQLAUTOCOMMIT is set to Yes:

```
SQLAUTOCOMMIT Yes              ; Paradox saves automatically (default)
ADD "Dlysales" "Mthsales"      ; after this operation
ADD "Dlysales" "Ytdsales"      ; and after this operation
EMPTY "Dlysales"               ; and after this operation
```

The following procedure performs two operations on the database. Assume that all the tables are remote. The two operations succeed or fail together, using the error procedure to control this process. (The error procedure used in this example is shown in the "Transaction processing" section of this chapter.)

```
PROC DoTransactionWithQuit()
  ErrorProc = "AlwaysRollbackAndQuit"
  SQLAUTOCOMMIT No             ; turn Paradox commits off
  SQLSTARTTRANS                ; server might require explicit
  IF Retval                    ; transaction start
    THEN
      ADD "WeekOrd" "Orders"
      EMPTY "WeekOrd"
      SQLCOMMIT                ; all operations succeeded
                               ; commit the transaction
    SQLAUTOCOMMIT Yes          ; turn Paradox commits on again
  ENDIF
  RETURN Retval                ; return Retval for evaluation
ENDPROC
```

See also

- SQLCOMMIT, SQLROLLBACK, and SQLSTARTTRANS

- the discussion of transaction processing earlier in this chapter and in Chapter 2
- Tools | SQL | Preferences | AutoCommit in Chapter 5 of this manual
- your server-specific addendum

SQLBREAKCONNECT

Breaks the current server connection.

Syntax

SQLBREAKCONNECT

Description

Like the equivalent Tools | SQL | Connection | Break menu command, the abbreviated PAL command SQLBREAKCONNECT rolls back any open transactions and disconnects from the current server. If you're not currently connected to a server, SQLBREAKCONNECT has no effect. SQLBREAKCONNECT does not alter the current connection parameters.

Use

In general, since any active workstation session requires some resources from the database server, you can minimize the time spent connected to it by using SQLBREAKCONNECT. By disconnecting at the end of a session, or in the middle of an application when you don't need to be connected, you can maximize the efficiency of your server. Use SQLMAKECONNECT to reconnect to the server after breaking the connection.

Important If SQLAUTOCOMMIT is set to No and you want to save your changes, you must explicitly commit your changes on the database server before breaking the current server connection. Otherwise, the database server will roll back the changes when Paradox breaks the connection.

Example

This example causes your workstation to disconnect from the server. The connection parameters are not reset.

```
SQLSELECTCONNECT     ; let user select a connection
  IF Retval          ; user connected
    THEN
      DoSQLWork()     ; do some SQL work and commit if necessary
      SQLBREAKCONNECT ; finished, don't need connection function does
      DoPdoxWork()    ; NOT use SQL features test to see if user can
      SQLMAKECONNECT  ; still connect to server
                      ; Note: You can let Paradox connect to a server
                      ; automatically, you don't have to use SQLMAKECONNECT.
      IF Retval
        THEN DoMoreSQLWork()
        ELSE QUIT "Cannot connect to server"
      ENDIF
      SQLBREAKCONNECT
    ELSE QUIT "Canceled by user"
  ENDIF
```

See also

- SQLCLEARCONNECT, SQLCOMMIT, SQLISCONNECT(), and SQLMAKECONNECT
- Tools | SQL | Connection | Break in Chapter 5 of this manual

SQLCLEARCONNECT

Removes current connection parameters, such as user name and password.

Syntax

SQLCLEARCONNECT

Description

Like the equivalent Tools | SQL | Connection | Clear menu command, the abbreviated PAL command SQLCLEARCONNECT clears the current server connection, the workspace, and all remote user names and passwords. SQLCLEARCONNECT does not break the server connection.

Use

SQLCLEARCONNECT tells Paradox that all menu operations are local without actually breaking the current connection. Until you select another server connection, Paradox will not try to access remote data with selected menu operations. If you select a replica with a different connection, Paradox then disconnects and rolls back any uncommitted changes. If you select a replica with the same connection, Paradox uses the existing connection to reconnect. In either case, Paradox prompts you for connection parameters.

SQLCLEARCONNECT is useful when you want to make sure that you are completely disconnected from the current server, prior to changing connections. It is also useful for removing your user name and password from memory, like the UNPASSWORD PAL command or the Tools | Protect | Clear | Password menu choice for Paradox tables.

Example

This example clears the current connection if it is not for the Microsoft or SYBASE SQL Server and lets the user select the appropriate connection.

```
IF SQLCONNECTINFO("Product") <> "MSSQL"
   THEN
      SQLCLEARCONNECT                   ; clear current connection
      SQLSELECTCONNECT                  ; select a new connection
ENDIF
```

See also

- ❒ SQLBREAKCONNECT, SQLISCONNECT(), and SQLMAKECONNECT
- ❒ Tools | SQL | Connection | Clear in Chapter 5 of this manual
- ❒ UNPASSWORD PAL command
- ❒ Password command in Chapter 17 of the Paradox *User's Guide*

SQLCOMMIT

Commits all open changes on the database server.

Syntax

SQLCOMMIT

Description

Like the equivalent Tools | SQL | Transaction | Commit command, the abbreviated PAL command SQLCOMMIT tells the database server to commit changes at the conclusion of a successful remote operation. SQLCOMMIT sets *Retval* to True if the attempt to commit the transaction succeeds, or to False if it fails (if, for example, Paradox is not connected to a database server).

Important If SQLAUTOCOMMIT is set to No and you want to save your changes, you must explicitly commit your changes on the database server before breaking the current server connection. Otherwise, the database server will roll back your changes when Paradox breaks the connection.

For an introduction to transaction processing concepts, see the discussion in Chapter 2 and the "Transaction processing" section earlier in this chapter.

Use

You need to issue an explicit commit for transactions only when SQLAUTOCOMMIT is set to No or when the command is enclosed in the SQL...ENDSQL command. This way, you can test to determine that an operation (or series of operations) was successful before saving your changes. Use SQLCOMMIT before breaking a server connection if you want to save your changes; any changes that haven't been committed when you break the connection are rolled back.

Example

In the following example, an error-handling procedure resets a PAL variable called SQLOk. SQLOk is used as a flag to stop the application from processing the current transaction. The error procedure shouldn't exit the application because you want to continue and recover from the error.

```
PROC SQLOkErrorProc()
   IF (ERRORCODE() < 1000)
      THEN QUIT ERRORMESSAGE()    ; any non-SQL error is fatal
   ENDIF
   SQLOk = False                  ; flag error to application
   RETURN 1                       ; just continue
ENDPROC
```

In the following example, the SQLOk flag is checked after each operation and determines whether the transaction should continue or be rolled back. (This example shows nested IFs, but you could also have the IF statements directly follow one another.)

```
PROC DoTransactionWithFlag()
  ErrorProc = "SQLOkErrorProc"
  SQLOk = True
  SQLSTARTTRANS                   ; server might require
                                  ; explicit transaction start
  IF Retval AND SQLOk
    THEN
      SQLAUTOCOMMIT No            ; turn off Paradox commits
      ADD "WeekOrd" "Orders"
      IF SQLOk
        THEN EMPTY "WeekOrd"
          IF SQLOk
            THEN SQLCOMMIT        ; commit the transaction
          ENDIF
      ENDIF
      SQLAUTOCOMMIT Yes           ; turn on Paradox commits again
  ENDIF
  IF (NOT Retval OR NOT SQLOk)
    THEN SQLROLLBACK RETURN False
  ENDIF
  RETURN True
ENDPROC
```

See also

- ❐ SQLAUTOCOMMIT, SQLROLLBACK, and SQLSTARTTRANS
- ❐ the discussion of transaction processing earlier in this chapter and in Chapter 2
- ❐ Tools | SQL | Transaction | Commit in Chapter 5 of this manual

SQLCONNECTINFO ()

Returns connection information about a remote table or from the current connection.

Syntax

SQLCONNECTINFO (*OptionName* [, *ReplicaName*])

OptionName is one of the following strings. (You can use any combination of uppercase and lowercase letters.)

"TITLE"	Returns the title of the connection (as displayed when you highlight a remote table, or when you select a connection).
"DESCRIPTION"	Returns the description of the connection.
"PRODUCT"	Returns the code name of the associated server product (see your server-specific addendum).
"DIALECT"	Returns the code name for the SQL dialect (see your server-specific addendum).

ReplicaName is the name of a replica for a remote table, which tells SQLCONNECTINFO() to return the connection information for that remote table.

Description

SQLCONNECTINFO() returns connection information about a remote table or the current connection. If you specify *ReplicaName*, SQLCONNECTINFO() returns information about that replica. Otherwise, it returns information on the current connection.

SQLCONNECTINFO() returns an empty (" ") string if the table you specify is not a replica, or if you don't specify a table name and the connection has not been set.

Use

Use SQLCONNECTINFO() to verify that a user has selected an appropriate connection, or to retrieve the connection description and display it.

Example

This example tests that a connection has been selected:

```
conntitle = SQLCONNECTINFO("TITLE")
IF conntitle = ""
   THEN MESSAGE "Connection is currently not set."
   ELSE MESSAGE "Connection is ", conntitle
ENDIF
```

The following example verifies that the dialect of the selected server connection is compatible with the SQL statements used in SQL...ENDSQL (that is, SQL statements written for the IBM Extended Edition dialect):

```
WHILE True
  SQLSELECTCONNECT                    ; select a connection
  conndial = SQLCONNECTINFO("DIALECT")
  IF conndial = "IBMEE"               ; right connection
    THEN
      canusethis = "Y"                ; can continue
      QUITLOOP
    ELSE MESSAGE "Cannot use selected connection for this application"
    SLEEP 3000
    MESSAGE "Try again (Y/N)?"
    ACCEPT "A1" TO Response
    IF NOT Retval
```

```
      THEN RETURN
    ENDIF
    IF UPPER (Response) <> "Y"
      THEN
        canusethis = "N"
        QUITLOOP
    ENDIF
  ENDIF
ENDWHILE
IF canusethis = "N"
  THEN RETURN "Will not continue with remote operations"
ENDIF
```

See also

❑ SQLISREPLICA() and SQLMAPINFO()

SQLERRORCODE ()

Returns the server code of the most recent error on the database server.

Syntax

SQLERRORCODE ()

SQLERRORCODE() takes no arguments. It returns a string indicating the most recent error on the database server, or an empty (" ") string if no error has occurred. If the database server produces a numeric value instead of a string, SQLERRORCODE() converts the number to a string.

Description

SQLERRORCODE() lets you see what kind of run-time error or error condition has occurred on the database server. In conjunction with the global variable *Errorproc*, SQLERRORCODE() lets you build an error-handling routine for remote operations into your application. SQLERRORCODE() reports on remote errors only; use ERRORCODE() to trap errors that occur in the local Paradox environment.

SQLERRORCODE() returns the server-specific error code. In other words, for the same type of error, the value returned on one database server product may differ from that of another. See your server manuals for more information about the error codes for your particular server and your server-specific addendum for an example.

Use

Use SQLERRORCODE() to create an error-handling routine for remote operations within your application. For examples and more information, see the discussion in "Error handling" earlier in this chapter.

See also

❑ ERRORCODE(), ERRORMESSAGE(), and SQLERRORMESSAGE()

- the discussion of error handling earlier in this chapter and in Chapter 7 of the *PAL Programmer's Guide*
- ERRORCODE() function in the *PAL Reference*
- your server-specific addendum

SQLERRORMESSAGE ()

Returns the message of the most recent run-time error.

Syntax

SQLERRORMESSAGE ()

SQLERRORMESSAGE() takes no arguments. It returns a string containing the text of the most recent run-time error or error condition on the database server, or a blank ("") string if no error has occurred.

Description

SQLERRORMESSAGE() is used to retrieve the message generated by the server when the last error occurred. SQLERRORMESSAGE() reports on remote errors only. Use ERRORMESSAGE() to return the message text of a local error.

Use

Use SQLERRORMESSAGE() to create an error-handling routine for remote operations within your application. See the discussion of "Error handling" earlier in this chapter, and your server-specific addendum for an example.

See also

- ERRORCODE(), ERRORMESSAGE(), and SQLERRORCODE()
- the discussion of error handling earlier in this chapter and in Chapter 7 of the *PAL Programmer's Guide*
- ERRORMESSAGE() function in the *PAL Reference*
- your server-specific addendum

SQLFETCH

Fetches one record at a time and copies it to an array.

Syntax

SQLFETCH *ArrayName*

Description

SQLFETCH is used in conjunction with the SQL NOFETCH...ENDSQL construct. You must use the NOFETCH keyword in SQL...ENDSQL before using this command. NOFETCH tells Paradox to let the PAL programmer choose when to fetch the result. SQLFETCH fetches one record at a time and copies it to a specified array. The first element of the array is not used (and is assigned a null " " string). The rest of the array is filled with the field values from your SELECT statement in the same format as COPYTOARRAY.

Retval is set to True if SQLFETCH returns a record and False if it does not.

Use

Use SQLFETCH in the following situations:

- when you need only one value (or a small set of values) returned from a remote query (when not all *Answer* results are needed)
- when you're in Edit or CoEdit mode and you want to insert a remote value into a local table, or do lookups on remote tables while you're editing
- when you want to show the results after fetching only the first few records, without waiting for all records to be collated in the *Answer* table
- when you want more control over your remote query results

You must use SQLRELEASE after your last SQLFETCH operation to release the pending query.

Note You cannot reference the field names directly as you can in COPYTOARRAY; you can only reference the field positions (1 through *ArraySize*(*ArrayName*)).

Example

This procedure returns the number of records in a table passed as a parameter:

```
PROC GetRecCount(Tbl);         ; returns number of records in table
   PRIVATE Rec
   SQL NOFETCH
      SELECT COUNT(*)
      FROM ~Tbl~
   ENDSQL
   SQLFETCH Rec                ; fetch single record into array
   SQLRELEASE                  ; release the pending query
   RETURN Rec[2]               ; second array element is the first
                               ; field returned by the query
ENDPROC
```

See also

- SQL...ENDSQL and SQLRELEASE
- COPYFROMARRAY and COPYTOARRAY in the *PAL Reference*

SQLISCONNECT ()

Determines whether Paradox is currently connected to a server.

Syntax

SQLISCONNECT ()

Description

SQLISCONNECT() tells whether you're currently connected to a server, and returns True if you are or False if you're not.

Use

Use SQLISCONNECT() to see if a server connection is active.

Example

This example shows how to test for a server connection before attempting to commit or roll back a transaction:

```
PROC EndTrans(DoCommit)
IF (SQLISCONNECT())          ; connected to a server
   THEN
      IF (DoCommit)
         THEN SQLCOMMIT
         ELSE SQLROLLBACK
      ENDIF
      IF (NOT Retval)
         THEN QUIT "Commit Failed: " + ERRORMESSAGE()
      ENDIF
   ELSE                      ; not connected to a server
   QUIT "Not connected to a server"
ENDIF
ENDPROC
```

The next example shows a procedure you could use to establish server connections:

```
PROC ConnectStatus()
IF SQLISCONNECT() THEN
   msg = "connected to server"
ELSE
   IF NOT SQLCONNECTINFO("TITLE") THEN
      SQLMAKECONNECT
      IF RetVal
         THEN msg = "Reconnected to server"
      ELSE
         msg = "Can't connect to server"
      ENDIF
   ELSE
      msg = "No connection specified"
   ENDIF
ENDIF
RETURN msg
ENDPROC
```

See also

❒ SQLBREAKCONNECT, SQLCLEARCONNECT, SQLMAKECONNECT, and SQLSELECTCONNECT

SQLISREPLICA ()

Finds out whether a table is a local table or a replica.

Syntax

SQLISREPLICA (*TableName*)

TableName is a string expression containing a Paradox table name.

Description

SQLISREPLICA() returns True if *TableName* is a replica, or False if it's not. SQLISREPLICA() also returns False if the specified *TableName* doesn't exist or is not a valid table or replica.

Use

You can use SQLISREPLICA() to verify that a table is a replica before attempting to use it for a remote operation on the server or before using SQLMAPINFO() or SQLCONNECTINFO(), described elsewhere in this chapter.

Example

This example tests whether a table is remote before renaming it:

```
IF ISTABLE("Orders")
  THEN
    IF SQLISREPLICA("Orders")
      THEN
        COPY "Orders" "LocOrd"
        DELETE "Orders"
      ELSE RENAME "Orders" "LocOrd"
    ENDIF
ENDIF
```

See also

❒ SQLCONNECTINFO() and SQLMAPINFO()

SQLMAKECONNECT

Attempts to reconnect to the server specified by the current connection.

Syntax

SQLMAKECONNECT

Description

Like the equivalent Tools | SQL | Connection | Make menu command, the abbreviated PAL command SQLMAKECONNECT attempts to reconnect to the server using the current connection. If a connection is not currently resident in memory, a script error results.

SQLMAKECONNECT sets *Retval* to True if the connection succeeds, or to False if it fails. If the connection fails, you can retrieve the reason for failure by using the error-handling functions

ERRORCODE(), ERRORMESSAGE(), SQLERRORCODE(), and SQLERRORMESSAGE().

Important You must explicitly commit your changes on the database server before making a new connection if you want to save your changes. Otherwise, the database server automatically rolls back the changes when Paradox breaks the connection.

Use

Use SQLMAKECONNECT to reconnect to the server after breaking the connection in one of the following ways:

- by selecting a different server connection
- by pressing *Ctrl-Break* during a remote operation
- by executing the PAL DOSBIG or RUN BIG commands
- by using *Alt-0* DOSBig
- by exiting Paradox

Paradox automatically attempts to reconnect to the server after your connection is broken at

- the first PAL or command menu that accesses a replica
- the first SQL...ENDSQL command
- the next SQLSTARTTRANS

Note SQLBREAKCONNECT attempts to reconnect to the global connection if you have both a main connection and a replica connection.

Example

This example uses SQLMAKECONNECT to test a broken connection before reconnecting:

```
SQLSELECTCONNECT      ; let user connect to server
  IF Retval           ; if Retval True, then connection has succeeded
    THEN
      DoSQLWork()     ; do some SQL work and commit if necessary
      SQLBREAKCONNECT ; finished, don't need connection anymore

      DoPdoxWork()    ; this function does not use SQL features
      SQLMAKECONNECT  ; test to see if user can still connect to server
                      ; Note: You can let Paradox connect to a server,
                      ; you don't have to use SQLMAKECONNECT.
      IF Retval
        THEN DoMoreSQLWork()
        ELSE QUIT "Cannot connect to server"
      ENDIF
      SQLBREAKCONNECT
    ELSE QUIT "Canceled by user"
  ENDIF
```

See also

- SQLBREAKCONNECT, SQLCLEARCONNECT, SQLISCONNECT(), SQLSELECTCONNECT, and SQLSETINTERRUPT

❐ Tools | SQL | Connection | Make in Chapter 5 of this manual

SQLMAPINFO ()

Returns structural information about a remote table.

Syntax

SQLMAPINFO (*OptionName, TableName* [, *ColumnNumber*])

OptionName is one of the following strings (you can use any combination of uppercase and lowercase letters):

"TABLENAME"	Returns the remote table name.
"COLUMNNAME"	Returns the name of the remote column specified by *ColumnNumber.*
"COLUMNTYPE"	Returns the type of the remote column specified by *ColumnNumber.*

TableName is a replica name.

ColumnNumber is the ordinal number of the field in the remote table; it is required if you specify "COLUMNNAME" or "COLUMNTYPE" for *OptionName.*

Description

SQLMAPINFO() gives you information about the structure of a remote table from its replica. This is useful if you want to work directly with the table using the SQL...ENDSQL command. SQLMAPINFO() results in an error if you specify an invalid table name, if the table you specify is not a remote table, or if the column number you specify is out of range.

Use

Use SQLMAPINFO() to return structural information about a remote table.

Example

This example retrieves the name of the third column in the remote table *Orders,* assigns that name to the variable *Getfield,* then substitutes that name in the SQL statement enclosed within the SQL...ENDSQL command:

```
Getfield = SQLMAPINFO("ColumnName","Orders",3)
SQL
   SELECT * FROM Orders
   WHERE ~Getfield~ = 0
ENDSQL
```

You can also do it this way:

```
SQL
   SELECT * FROM Orders
   WHERE ~SQLMAPINFO("ColumnName","Orders",3)~ =
                                      ; uses name of third column
ENDSQL
```

The following example fills an array with all field names and field types:

```
DYNARRAY Rtable
FOR i FROM 1 to NFIELDS("ORDERS")
  Rtable [SQLMAPINFO("ColumnName","ORDERS",i)]=
    SQLMAPINFO("ColumnType", "ORDERS", i)
ENDFOR
```

See also

❒ SQLCONNECTINFO() and SQLISREPLICA()

SQLRELEASE

Releases the pending query.

Syntax

SQLRELEASE

Description

SQLRELEASE is used after the SQLFETCH command to prepare Paradox for another query. You *must* use SQLRELEASE after each SQL NOFETCH...ENDSQL command.

Use

Use SQLRELEASE to release the resources used for the last SQLFETCH. If you do not use SQLRELEASE to release the pending query, a script error results when you attempt another query.

Example

This procedure returns the number of records in a table passed as a parameter:

```
PROC GetRecCount(Tbl)
   PRIVATE Rec
   SQL NOFETCH
      SELECT COUNT(*)
      FROM ~Tbl~
   ENDSQL
   SQLFETCH Rec       ; fetch single record into array
   SQLRELEASE         ; release the pending query
   RETURN Rec[2]
ENDPROC
```

See also

❒ SQL...ENDSQL and SQLFETCH

SQLRESTORECONNECT

Restores a connection saved with SQLSAVECONNECT.

Syntax **SQLRESTORECONNECT**

Description SQLRESTORECONNECT restores the connection saved by the most recent use of SQLSAVECONNECT. (Once you restore a connection, it clears the stored connection from memory.) SQLRESTORECONNECT also clears the current connection from memory, so you must first issue another SQLSAVECONNECT to save connections if you may need them again. In this case, the last-saved connection in memory is overwritten. If no setting has been saved, SQLRESTORECONNECT clears the current connection.

Use Use SQLRESTORECONNECT to restore a saved connection.

Example This example restores the connection saved with SQLSAVECONNECT:

```
SQLSAVECONNECT              ; saves current connection
SQLSELECTCONNECT            ; select new connection
IF Retval                   ; user selected a connection
   THEN Do_Work()           ; do work on different server
ENDIF
SQLRESTORECONNECT           ; return to previous connection
```

See also

- ❒ SQLSAVECONNECT and SQLSELECTCONNECT
- ❒ Tools | SQL | Connection | Select in Chapter 5 of this manual

SQLROLLBACK

Rolls back changes on the database server.

Syntax **SQLROLLBACK**

Description Like the equivalent Tools | SQL | Transaction | RollBack menu command, the abbreviated PAL command SQLROLLBACK tells the database server to roll back current changes at the conclusion of an unsuccessful remote operation or transaction. If SQLAUTOCOMMIT is set to Yes, Paradox automatically commits your remote changes immediately after every operation so you cannot roll back changes.

If SQLAUTOCOMMIT is set to No, you must commit or roll back remote changes yourself. Paradox never commits changes resulting from SQL...ENDSQL operations, so you must always commit or roll back these changes yourself. SQLROLLBACK abandons changes to remote tables but not to local tables; Paradox always saves local table changes automatically.

SQLROLLBACK sets *Retval* to True if the attempt to roll back the transaction succeeds, or to False if it fails (if, for example, Paradox is not connected to a database server). This command lets you test if the rollback succeeded in the script and traps the error if it did not.

You will get a script error if you use SQLROLLBACK and you have not specified a connection.

Important If SQLAUTOCOMMIT is set to No, you must explicitly commit your changes on the database server before breaking the current connection. Otherwise, the database server will roll back your changes when Paradox breaks the connection.

Note If your server does not automatically start a transaction and SQLAUTOCOMMIT is set to No, you must use SQLSTARTTRANS if you want to be able to roll back changes on that server (see your server-specific addendum for more information).

For an introduction to transaction processing concepts, see the discussion in Chapter 2 and the "Transaction processing" section earlier in this chapter.

Use

SQLROLLBACK lets you abandon changes to remote tables if an operation (or series of operations) was not completed successfully. This ensures the maintenance of data integrity in the event of an error.

Example

This example shows how to roll back changes if an error occurs during a transaction. You don't need to set SQLAUTOCOMMIT to No because the only SQL statement in this example is enclosed in a SQL...ENDSQL command, which Paradox never automatically commits.

```
PROC SetSQLError()
  PdoxCode = ERRORCODE()
  IF(PdoxCode >= 1000)
    THEN RETURN 1              ; SQL errors are left for
                               ; application to handle
    ELSE QUIT ERRORMESSAGE()   ; any other error is fatal
  ENDIF
ENDPROC

PROC DoTransaction()
  ErrorProc = "SetSQLError"
  PdoxCode = 0
  SQLSTARTTRANS                ; server might require explicit
                               ; transaction start
  IF (Retval AND PdoxCode = 0)
    THEN VIEW "LocCust"
      SCAN FOR [CustName] = "John Smith"
      SQL
      INSERT INTO Orders (Ordnum, Quantity)
      VALUES (~[OrdNum]~,~[Quantity]~)
    ENDSQL
    IF PdoxCode <> 0
      THEN                     ; error procedure called
        SQLROLLBACK            ; roll back and return
        RETURN False
    ENDIF
    ENDSCAN
  SQLCOMMIT
ENDIF
RETURN Retval
ENDPROC
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, and SQLSTARTTRANS
- the discussion of transaction processing earlier in this chapter and in Chapter 2
- Tools | SQL | Transaction | RollBack in Chapter 5 of this manual

SQLSAVECONNECT

Temporarily saves the current connection.

Syntax

SQLSAVECONNECT

Description

SQLSAVECONNECT temporarily saves the current connection in memory. This enables you to select and use other connections, then return to this saved connection (using SQLRESTORECONNECT) when you're through. SQLSAVECONNECT overwrites any connection saved with a previous SQLSAVECONNECT. If there is no connection, SQLSAVECONNECT has no effect.

Use

In an application, you'll usually save a connection with SQLSAVECONNECT before you connect to another server. When you have completed your work on the other server, you can use SQLRESTORECONNECT to return to the previous connection.

Example

This example saves your connection while you use another server connection:

```
SQLSAVECONNECT              ; save current connection
SQLSELECTCONNECT            ; select different connection
IF Retval                   ; user selected a connection
   THEN DoWork()            ; do work on different server
   ELSE MESSAGE "Current connection not changed"
ENDIF
SQLRESTORECONNECT           ; restore previous connection
```

See also

- SQLRESTORECONNECT and SQLSELECTCONNECT
- Tools | SQL | Connection | Select in Chapter 5 of this manual

SQLSELECTCONNECT

Selects a server connection.

Syntax

SQLSELECTCONNECT [{ **PRODUCT** *ProductName* | **TITLE** *Title* } **VALUES** *ParameterList*]

If you use SQLSELECTCONNECT with no parameters, Paradox displays the Set Connections screen (a list of available connections to choose from), and then prompts for the appropriate connection parameters. *Retval* is set to True if the user chooses a connection and to False if the user cancels without choosing a connection.

You can use either the PRODUCT or TITLE keywords to indicate the connection. *ProductName* is the code name of the server product (see your server-specific addendum). *Title* is the connection name shown on the SQL Connections screen (see Chapter 7).

ParameterList is a comma-separated list of the connection parameters required for this connection (see your server-specific addendum). Connection parameters must be entered in the same sequence as they appear on the SQL Connections screen (see Chapter 7).

Description

Like the equivalent Tools | SQL | Connection | Select menu command, the abbreviated PAL command SQLSELECTCONNECT lets you select a server connection. Paradox uses this connection for all operations in the current session (until you select another connection).

Use

For certain operations (like SQL...ENDSQL, CREATE REMOTE, SQLSTARTTRANS, SQLROLLBACK, and SQLCOMMIT), you need to connect to the server first; SQLSELECTCONNECT lets you do so. You can test the selected connection for compatibility with your application by using SQLCONNECTINFO().

For other commands, you can make the connection by using a replica and entering the user name and password, if required. You must explicitly connect to the server if the first operation your application performs

- creates a replica (like CREATE REMOTE)
- uses SQL passthrough (like a SQL...ENDSQL command)

Example

This example prompts the user to choose a connection:

```
SQLSELECTCONNECT           ; select a connection
   IF Retval               ; user selected a connection
      THEN DoWork()        ; do work on this server
      ELSE MESSAGE "Connection not selected"
   ENDIF
```

This example sets the connection for the Microsoft or SYBASE SQL Server:

```
SQLSELECTCONNECT PRODUCT "MSSQL"        ; select a server connection
VALUES "mary", "null", "mis", "cust"    ; user name, password, server,
                                        ; and database
```

See also

- SQLBREAKCONNECT, SQLCLEARCONNECT, SQLCONNECTINFO(), SQLMAKECONNECT, SQLRESTORECONNECT, and SQLSAVECONNECT
- Tools | SQL | Connection | Select in Chapter 5 of this manual
- Chapter 7 of this manual for information on SQL Setup

SQLSETINTERRUPT

Determines whether Paradox interrupts a remote PAL operation when the user presses *Ctrl-Break*.

Syntax

SQLSETINTERRUPT { Yes | No }

Description

Like the equivalent Tools | SQL | Preferences | SetInterrupt menu command, the abbreviated PAL command SQLSETINTERRUPT tells Paradox whether to let the user interrupt the current PAL remote operation. If SQLSETINTERRUPT is Yes (the default), Paradox interrupts the operation when the user presses *Ctrl-Break*, rolls back any open transactions, breaks the server connection, and returns an error message (error code 1003). If SQLSETINTERRUPT is No and the user presses *Ctrl-Break*, Paradox waits for the remote command to finish executing, then halts the script.

Use

By default, SQLSETINTERRUPT is set to Yes, which lets the user interrupt a remote operation. This can be useful, for example, if a user attempts an operation on a remote table that is locked or running a long query. A user can also use *Ctrl-Break* if they want to stop a query that is returning too many records.

You would set SQLSETINTERRUPT to No during an operation or series of operations that you wanted to complete without having to handle all the cases where the user might interrupt it.

Example

This example shows you how you can control the effect of pressing *Ctrl-Break* while adding data between two remote tables:

```
SQLSETINTERRUPT Yes           ; user can interrupt
ADD "Orders" "Dlysales"
SQLSETINTERRUPT No            ; user cannot interrupt
EMPTY "Orders"
```

See also

- Tools | SQL | Preferences | SetInterrupt in Chapter 5 of this manual
- CTRLBREAK command in the *PAL Reference*

SQLSTARTTRANS

Starts a transaction on the database server.

Syntax

SQLSTARTTRANS

Description

Like the equivalent Tools | SQL | Transaction | Start menu command, the abbreviated PAL command SQLSTARTTRANS begins a transaction on the database server. All subsequent operations are grouped in the current transaction until they are committed (using SQLCOMMIT) or rolled back (using SQLROLLBACK).

SQLSTARTTRANS sets *Retval* to True if the transaction begins successfully, and to False if it fails (for example, if Paradox is not currently connected to a server). This lets you test for successful transactions in the script and traps the error if the transaction failed.

Note You might not need to use this command. Some database servers require that you explicitly start a new transaction when you commit or roll back operations, while other servers automatically start a new transaction for you. You must use SQLSTARTTRANS to begin a transaction on those database servers that require you to do so explicitly. See your server-specific addendum to find out how your database server manages transactions.

Once you have committed or rolled back a transaction, you might need to issue SQLSTARTTRANS to begin the next transaction.

Use

If your server does not automatically start a transaction and SQL AUTOCOMMIT is set to No, you must use SQLSTARTTRANS if you want to be able to roll back changes on that server. If your database server starts a transaction automatically, SQLSTARTTRANS has no effect. It's best to include SQLSTARTTRANS in any application that might run on a server that requires it.

Example

This example shows how to start a transaction on a database server. The *SetSQLError* procedure sets *PdoxCode* to the value of the error code that occurs.

```
PROC DoTransaction()
  ErrorProc = "SetSQLError"
  PdoxCode = 0
  SQLSTARTTRANS               ; server might require explicit
                              ; transaction start
  IF (Retval AND PdoxCode = 0)
    THEN
      VIEW "LocCust"
      SCAN FOR [CustName] = "John Smith"
      SQL
        INSERT INTO Orders (OrdNum, Quantity)
        Values (~[OrdNum]~,~[Quantity]~)
      ENDSQL
      IF PdoxCode <> 0
        THEN SQLROLLBACK
        RETURN False
      ENDIF
      ENDSCAN
    SQLCOMMIT
  ENDIF
  RETURN Retval
ENDPROC
```

See also

- SQLAUTOCOMMIT, SQLCOMMIT, and SQLROLLBACK
- Tools | SQL | Transaction | Start in Chapter 5 of this manual

SQLVAL ()

Translates a PAL expression to a valid SQL expression (for use in SQL...ENDSQL statements).

Syntax

SQLVAL (*Expression*)

Description

SQLVAL() accepts a PAL expression and translates it to a SQL string as follows:

- If *Expression* evaluates to a blank value, SQLVAL() returns the string "NULL".
- If *Expression* is an alphanumeric value, SQLVAL() returns the string " ' " + Expression + " ' ".
- If *Expression* is an alphanumeric value and it contains one or more embedded single-quote characters, SQLVAL() delimits the quote characters.
- If *Expression* is a numeric (N, S, or $) value, SQLVAL() returns the string "Expression".

- If *Expression* is a date value, SQLVAL() returns a date expression that complies with the dialect of the current connection.

You must select a connection before using SQLVAL() because SQLVAL() uses the target SQL dialect from the current connection.

Use

Use this function to prepare a PAL expression for inclusion in SQL...ENDSQL commands. SQLVAL() handles type translation and blank values (which are translated to the keyword NULL), and translates PAL expressions into the appropriate SQL dialect.

Example

The following examples show how SQLVAL() translates expressions to the values that the server is expecting. The first example uses these variables:

```
Name = "John"
LastName = "Smith"
Date = TODAY()          ; say 10/06/90
Number = 3.5
Address = ""
Zip = BLANKNUM()
```

Given the following query,

```
SQL
  INSERT INTO RemTab
  VALUES(~SQLVAL(Date+1)~,~SQLVAL(Name)~,~SQLVAL(LastName)~,
         ~SQLVAL(Number*3)~,~SQLVAL(Zip)~,~SQLVAL(Address)~)
ENDSQL
```

and assuming that the current connection uses the ORACLE dialect, the following query will be sent to the server:

```
SQL
  INSERT INTO RemTab
  VALUES(TO_DATE('10/07/90','MM/DD/YYYY'), 'John', 'Smith',
         10.5,NULL,NULL)
ENDSQL
```

The second example shows how you can update data on the server by scanning the local table (*LClients*) and performing UPDATE queries (assuming no client has a blank ID):

```
VIEW "LClients"
SCAN
  SQL
    UPDATE Client
    SET Address = ~SQLVal([ADDRESS])~,ZipCode = ~SQLVal([ZIP])~
    WHERE ClientId = ~SQLVAL([ID])~
  ENDSQL
ENDSCAN
```

Given this *LClients* table,

Table 6-8 *LClients* table

ID (N)	Name (A25)	Address (A25)	Zip (N)
137	Rick Jones		
123	John Smith	1800 Market	94114
145	Steve Doe	123 'I' Street	95001

the following commands will be sent to the server:

```
UPDATE Client
SET Address = NULL, ZipCode = NULL
WHERE ClientId = 137

UPDATE Client
SET Address = '1800 Market', ZipCode = 94114
WHERE ClientId = 123

UPDATE Client
SET Address = '123 ''I'' Market', ZipCode = 95001
WHERE ClientId =145
```

Note that the 'I' character is enclosed in single quotes.

Memo fields While Paradox supports fields of type **M** (memo fields), as do many database servers, SQL Link does not. You cannot use SQL Link to create memo fields. If you view data from memo fields in remote tables, it appears as a field type A255.

See also

- SQL...ENDSQL
- BLANKDATE(), BLANKNUM(), ISBLANK(), and STRVAL() functions in the *PAL Reference*
- the discussion of nulls in Chapter 2 of this manual

CHAPTER 7

The SQL Setup program

This chapter tells you how to use the SQL Setup program to create replicas and to configure the Paradox environment to fit your particular needs.

When to use SQL Setup

SQL Setup extends Paradox SQL Link's functionality by letting you create replicas and add and customize server connections. While you can use SQL Link without ever running SQL Setup, you can access only remote tables created in Paradox. Use SQL Setup if you want to work with the following types of remote tables in Paradox:

- tables created with programs other than Paradox
- tables created with the PAL SQL...ENDSQL command or with UseSQL
- tables with replicas stored in a directory that you don't have access to (for example, a replica that is stored in another user's private directory or on another user's local drive)
- tables with obsolete replicas (for example, tables that have been restructured outside Paradox)

To work with one of these types of tables, you need to create a replica for the existing remote table. Paradox uses that replica to locate the table on the database server.

SQL Setup lets you use Paradox to work with SQL views. Like remote tables, you need to create a replica to access remote views; then you can treat them just as you would any other remote table.

SQL Setup also lets you add custom connections, modify existing connections, or delete connections that you no longer need. You can customize a connection for an individual user, for a group of users, or for a particular application. You can also password-protect each connection.

SQL Setup is available anywhere in Paradox by playing the *SQLSetup* script or by choosing ≡ | Utilities | SQL Setup.

Creating replicas

As discussed in Chapter 2, a replica is a special local Paradox table that represents a remote table on a database server. A replica contains the connection information Paradox needs to find the table on the server, and the structural information it needs to work with the data in the table.

When you use Paradox to create a new remote table (by choosing Create | Remote or Tools | Copy from the Paradox Main menu, or by issuing the CREATE or COPY command in a PAL program), Paradox automatically creates the corresponding local replica.

If you create a remote table in any other way, however, you need to use SQL Setup to create a replica so Paradox can access it. Unless the structure of (or connection to) a remote table changes, or the replica is deleted or moved beyond your access, you need to create its replica only once. (The connection to a remote table can change if the name of the server or your access and path to the server changes.)

Note SQL Setup always creates replicas in the current working directory. You must have full rights in the current directory and have exclusive use of the directory while creating replicas with SQL Setup.

Using views on wide tables The maximum Paradox record size for keyed (indexed) tables is 1350 bytes; for unkeyed (nonindexed) tables it's 4000 bytes. To replicate tables with larger record lengths, create a view on a subset of the fields in these tables. If tables can't be replicated, or must be renamed, SQL Setup creates a table called *SQLProbs* containing a list of the problem tables along with descriptions of the problems encountered.

Field names with spaces SQL Setup does not replicate tables whose field names contain spaces (for example, "LAST NAME"). If SQL Setup encounters spaces in column names, it puts the table name in the *SQLProbs* table.

After SQL Setup finishes, you can create a report showing tables that were replicated, along with their field names and field types.

Customizing server connections

SQL Link comes with a default server connection for each server it supports. A server connection identifies the type of remote server and its SQL dialect, and contains title and description information that Paradox displays when you select the connection or when you work with a remote table. In addition, each connection contains connection parameters, such as user name and server name, that the server requires for access. See your server-specific addendum for details of these connection parameters.

When you choose Tools | SQL | Connection | Select from the Paradox Main menu, or when you issue SQLSELECTCONNECT from a PAL application, Paradox displays a list of server connections. While you can use the default connections that come with SQL Link to communicate with your server, SQL Setup lets you add custom connections to this list to fit the needs of a single user, a group of users, or an application.

For example, you can use SQL Setup's Connection | Customize option to modify the title information for a particular connection so that Paradox displays a title like "Personnel Data" to users rather than the type of server. You can also supply the network name of the server so that users don't have to type it in each time. If you have more than one database server, you could set up custom connections to each server for each user or for a group of users.

SQL Setup function keys

You can use the following function keys almost anywhere in SQL Setup:

- *F1* Help displays help on how to use the options available from SQL Setup's Main menu.
- *F2* Do_It! saves changes and proceeds to the next step.
- *F10* Menu displays a menu for the current screen. Standard menu choices include
  - Help (the equivalent of pressing *F1* Help)
  - DO-IT! (the equivalent of pressing *F2* Do_It!)
  - Cancel abandons the current operation and returns to the previous menu

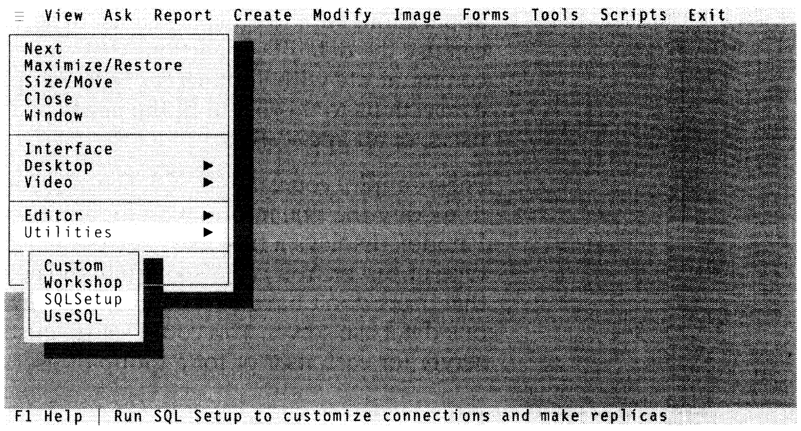
In addition, some screens have function keys for special operations that are applicable only in those situations. For example, in the Type Convert screen, you can press *Spacebar* to toggle between numeric and currency data types.

Before you start SQL Setup

If you plan to create replicas, make sure your working directory is the one where you want to save replicas. You must have write privileges in this directory and exclusive use of it (no other users can enter the directory or use any of the tables stored in it during your SQL Setup session). Tools | More | Directory changes your working directory.

Starting SQL Setup

You can start SQL Setup by choosing ≡ | Utilities | SQL Setup.


≡ View Ask Report Create Modify Image Forms Tools Scripts Exit
Next
Maximize/Restore
Size/Move
Close
Window
Interface
Desktop ▶
Video ▶
Editor ▶
Utilities ▶
Custom
Workshop
SQLSetup
UseSQL
F1 Help | Run SQL Setup to customize connections and make replicas

The Main menu options in SQL Setup are described in the following table:

Table 7-1 SQL Setup menu options

Menu	Description
Connection	Identifies and modifies the server connection.
MakeReplicas	Creates replicas for existing remote tables.
Help	Provides help about how to use the options in the SQL Setup Main menu.
Exit	Exits the SQL Setup Program and returns to the Paradox Main menu.

The rest of this chapter discusses how to work with each of the menus in the SQL Setup Program, and is organized by menu topic.

Connection

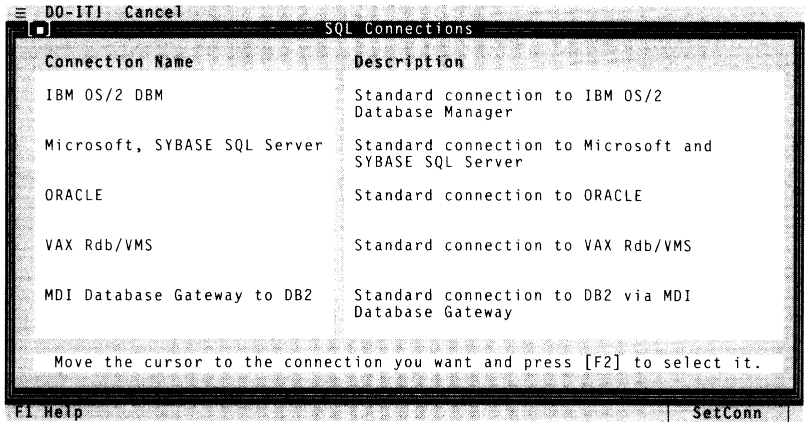
Once you establish a server connection, all options on the SQL Setup Main menu are available; otherwise, the option to create replicas is unavailable.

The SQL Setup Connection menu options are described in the following table:

Table 7-2 SQL Setup Connection menu options

Menu option	Description
Select	Specifies a server connection if you are not connected to a server. If you have a connection established but want to change it, you can use this option to break the current connection and establish a new one.
Make	Re-establishes the server connection stored in memory after you use Connection|Break or break the connection some other way.
Break	Breaks the current server connection without immediately establishing another connection. Information about the connection remains in memory.
Status	Displays one of three different messages indicating the status of a connection: 1. **Connected to *<servername>*** (if you're connected to a server) 2. **Connection set to *<servername>*** (if you've specified a server connection but aren't currently connected) 3. **Not connected to a server** (if you haven't specified a connection)
Customize	Creates a customized connection to a supported database server. Customized connections can have descriptive names and you can specify parameters that users would otherwise need to enter each time they wanted to connect to a server. You don't need to be connected to a server to use Customize.

If you didn't establish a connection prior to starting SQL Setup, you can use Connection | Select to do so. SQL Setup displays a list of existing connections and lets you select one of them, as shown in the following figure:



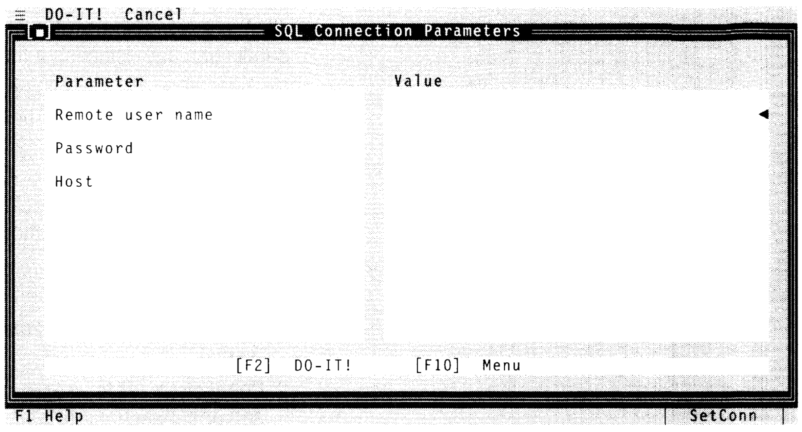
Connection Name	Description
IBM OS/2 DBM	Standard connection to IBM OS/2 Database Manager
Microsoft, SYBASE SQL Server	Standard connection to Microsoft and SYBASE SQL Server
ORACLE	Standard connection to ORACLE
VAX Rdb/VMS	Standard connection to VAX Rdb/VMS
MDI Database Gateway to DB2	Standard connection to DB2 via MDI Database Gateway

Move the cursor to the connection you want and press [F2] to select it.

Selecting a connection

To select a connection, press ↓ until you reach the desired connection, then press *F2* Do_It!.

A screen similar to the following one appears.



```
Parameter                     Value
Remote user name
Password
Host
```

Type the connection parameters required to access the appropriate database, then press *F2* Do_It!. For information about the connection parameters required for your server, consult your server-specific addendum. SQL Setup then diplays your new connection. Press *Enter* or click OK to close the dialog box.

Customizing a connection

Paradox stores customized connections in a file called PARADOX.DSQ. When you choose Connection | Customize from the SQL Setup Main menu, SQL Setup searches for the PARADOX.DSQ file in the current directory. If PARADOX.DSQ is not in the current directory, SQL Setup searches the Paradox system files directory, then continues to search along the DOS path. SQL Setup uses the first PARADOX.DSQ file it finds.

If SQL Setup can't find PARADOX.DSQ in any of these locations, it uses the default server connection for each supported product. Once you're through customizing a connection, you can save the file to any directory you choose on your local hard disk or network drive.

Caution Since Connection | Customize modifies the way Paradox users connect to a database server, access to a public PARADOX.DSQ should be reserved for authorized users only. Furthermore, because it might contain open passwords, it's advisable to password protect PARADOX.DSQ. Check with your database administrator before attempting to modify a public PARADOX.DSQ.

To customize your server connections, choose Connection | Customize from the SQL Setup Main menu. (If the PARADOX.DSQ file is password protected, SQL Setup prompts you for a password.) SQL Setup displays the SQL Connection screen.

Connection names, descriptions, and parameters

For each connection, you can specify (or change)

- The connection name for this connection. You can enter any text (up to 28 characters) as a title for this connection. This title appears with SQL Link messages and identifies replicas when you scroll through a table list.
- The description for this connection. You can enter more specific information about this connection (up to 80 characters).

  You can modify the connection name and description to say anything you want. This is the information that users see when they select a server connection from the connection list. Each connection name and description pair must be unique; you cannot have identical combinations.
- The parameters for the current connection. SQL Setup lists the parameters that SQL Link supplies to the database server to establish the connection, such as user name, password, and server name (you may want to fill in only the server name for users and ask them to supply their own user name and password). If all parameters (including password) are filled in, users will not see the parameters screen.

To edit connection data, use the keys listed in Table 7-3.

Table 7-3 Customize connection key operations

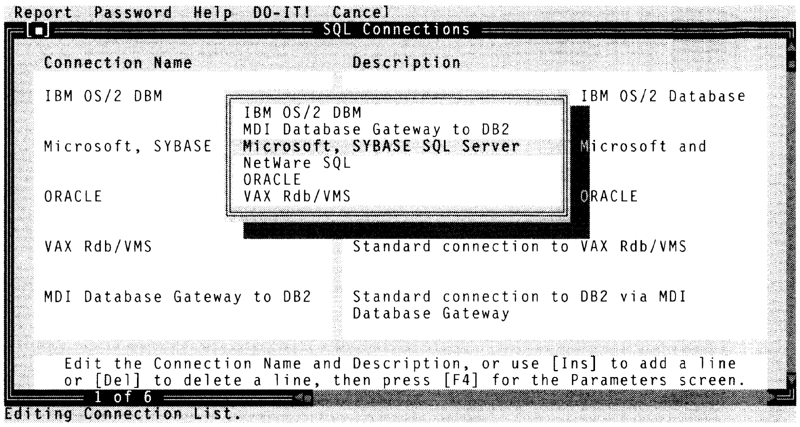
Key	Description
Home	Moves to first connection when in the Title area, or to first field when in the parameter area.
End	Moves to last connection when in the Title area, or to last field when in the parameter area.
↑	Moves to previous field or connection.
↓	Moves to next field or connection.
PgUp and *PgDn*	Scrolls among connections.
Ins	Adds a new connection at the current cursor position.
Del	Deletes the connection at the current cursor position.
F1 Help	Provides information about how to use options in the Connection menu.
F2 Do_It!	Validates your entries and saves any changes to PARADOX.DSQ.
F3 Move Up	Moves between title and parameter areas.

Key	Description
F4 Move Dn	Moves between title and parameter areas.
F10 Menu	Displays the Connections menu.

Adding a new connection

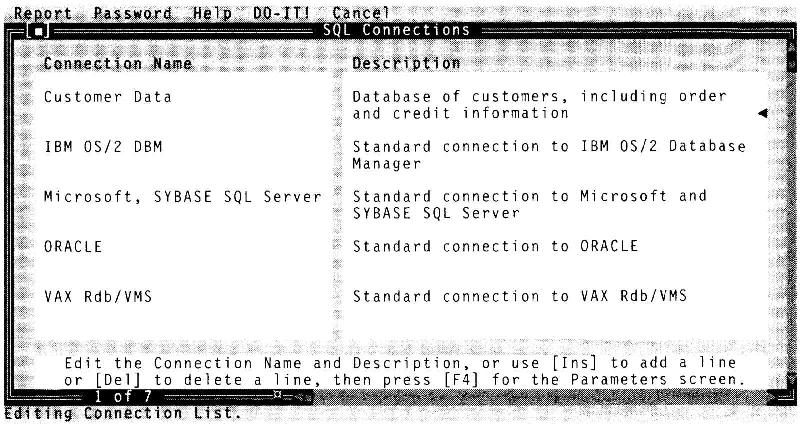
To add a new server connection, follow these steps:

1. Press *Ins* to insert a connection at the current cursor position. A list of available SQL products appears in the center of the screen. Choose the product (and dialect) on which you want to base the connection.



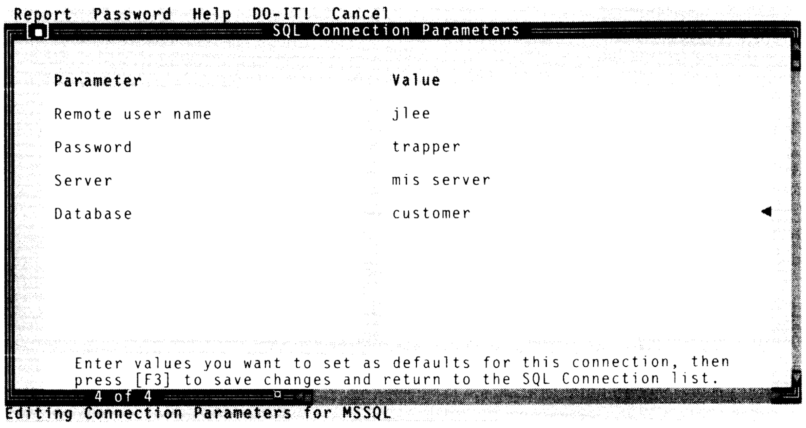
```
Report  Password  Help  DO-IT!  Cancel
                        SQL Connections

Connection Name                 Description

IBM OS/2 DBM                                          IBM OS/2 Database
                  IBM OS/2 DBM
                  MDI Database Gateway to DB2
Microsoft, SYBASE Microsoft, SYBASE SQL Server        Microsoft and
                  NetWare SQL
                  ORACLE
ORACLE            VAX Rdb/VMS                         ORACLE

VAX Rdb/VMS                     Standard connection to VAX Rdb/VMS

MDI Database Gateway to DB2     Standard connection to DB2 via MDI
                                Database Gateway

  Edit the Connection Name and Description, or use [Ins] to add a line
  or [Del] to delete a line, then press [F4] for the Parameters screen.
    1 of 6
Editing Connection List.
```

2. A new connection appears in the connection list. Edit the connection name and description as described in the previous section. Each connection name and description pair must be unique; you cannot have identical combinations.



```
Report  Password  Help  DO-IT!  Cancel
                        SQL Connections

Connection Name                 Description

Customer Data                   Database of customers, including order
                                and credit information

IBM OS/2 DBM                    Standard connection to IBM OS/2 Database
                                Manager

Microsoft, SYBASE SQL Server    Standard connection to Microsoft and
                                SYBASE SQL Server

ORACLE                          Standard connection to ORACLE

VAX Rdb/VMS                     Standard connection to VAX Rdb/VMS

  Edit the Connection Name and Description, or use [Ins] to add a line
  or [Del] to delete a line, then press [F4] for the Parameters screen.
    1 of 7
Editing Connection List.
```

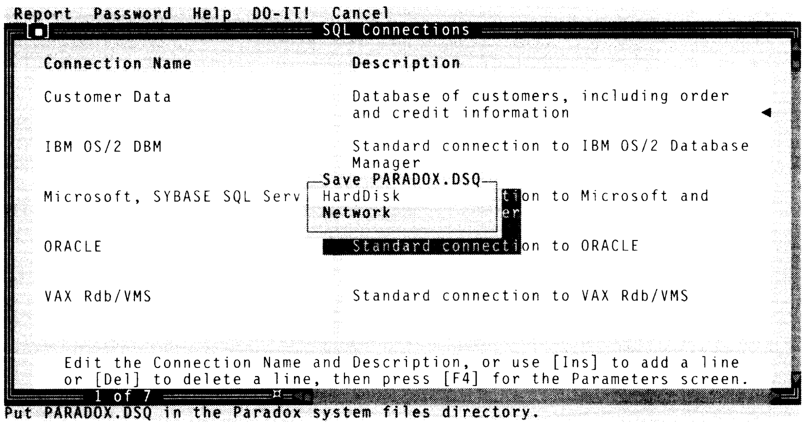
3. Press *F4* Move Dn to move to the SQL Connection Parameters screen. Edit the connection parameters as described in the previous section.



```
Report  Password  Help  DO-IT!  Cancel
                    SQL Connection Parameters

  Parameter                          Value

  Remote user name                   jlee

  Password                           trapper

  Server                             mis server

  Database                           customer                          ◄


    Enter values you want to set as defaults for this connection, then
    press [F3] to save changes and return to the SQL Connection list.
       4 of 4
Editing Connection Parameters for MSSQL
```

Saving connection data

When you press *F2* Do_It! on the SQL Connection screen, SQL Setup validates your entries, then asks you to decide where to save the connection data in PARADOX.DSQ:



```
Report  Password  Help  DO-IT!  Cancel
                         SQL Connections

  Connection Name                  Description

  Customer Data                    Database of customers, including order
                                   and credit information              ◄

  IBM OS/2 DBM                     Standard connection to IBM OS/2 Database
                                   Manager
                                 ┌Save PARADOX.DSQ┐
  Microsoft, SYBASE SQL Serv│ HardDisk         │tion to Microsoft and
                                 │ Network          │er
                                 └──────────────────┘
  ORACLE                           Standard connection to ORACLE

  VAX Rdb/VMS                      Standard connection to VAX Rdb/VMS

   Edit the Connection Name and Description, or use [Ins] to add a line
   or [Del] to delete a line, then press [F4] for the Parameters screen.
     1 of 7
Put PARADOX.DSQ in the Paradox system files directory.
```

- Hard Disk: Saves PARADOX.DSQ in the directory where it was found. If no previous version is found, PARADOX.DSQ is saved in the Paradox system files directory.
- Network: Saves PARADOX.DSQ to a directory you specify on your local hard disk or network drive. SQL Setup prompts you for the complete path.

Note You cannot save an empty list.

Once SQL Setup saves PARADOX.DSQ, you return to the SQL Setup Main menu.

Password protection for system administrators

Since PARADOX.DSQ can contain passwords, you may want to password protect it. There are two levels of password protection:

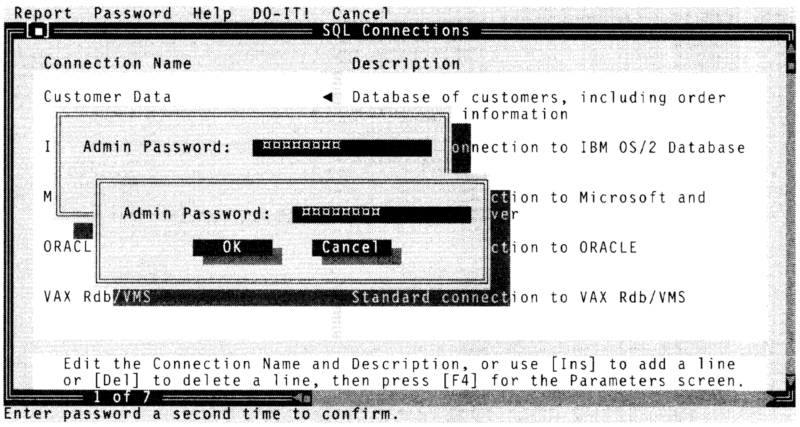
- Admin level: This password should be known only by the system administrator. Admin-level passwords are the same as owner passwords in Paradox.
- User level: All users share a user level, or auxilliary, password, which they must provide to use the connections list.

For more information, see Chapter 17 of the Paradox *User's Guide*.

Important If you password protect the connections list, any user who does not have a password will not be able to connect to a server.

To password protect the connections list, follow these steps:

1. Choose Password | SetAdmin from the Customize Main menu.
2. Enter the password and choose OK. Paradox will ask you to verify your entry by entering the password again.



```
Connection Name          Description

Customer Data          ◄ Database of customers, including order
                           information
I                        ...nection to IBM OS/2 Database
M                        ...ion to Microsoft and
                         ...r
ORACL                    ...ion to ORACLE
VAX Rdb/VMS              Standard connection to VAX Rdb/VMS

   Edit the Connection Name and Description, or use [Ins] to add a line
   or [Del] to delete a line, then press [F4] for the Parameters screen.
Enter password a second time to confirm.
```

If you want to add a user-level password, choose Password | SetUser from the Customize Main menu. User passwords are similar to auxiliary passwords in Paradox with the following exceptions:

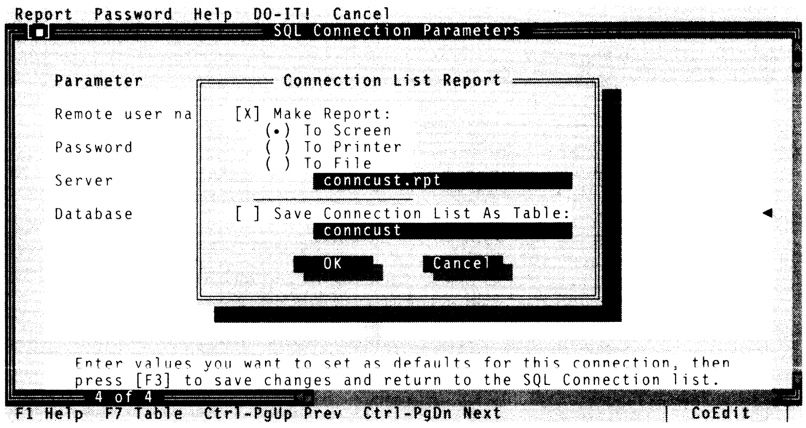
- You can have only one user-level password (which you can give to all authorized users) for each PARADOX.DSQ.
- User passwords grant the user read-only access to the list of customized connections.

Password protection for your personal PARADOX.DSQ

If you have a personal PARADOX.DSQ that you want to password protect, follow the instructions for setting up an admin-level password as explained previously. Although you don't need to define any user passwords, you can do so if you want other users to have access to your custom connections, but you don't want to let them change any connection information. You will need to enter your admin-level password each you want to use your personal PARADOX.DSQ.

Customize connection reports

To create a report on your session using Customize, choose Report from the Customize Main menu.


Report Password Help DO-IT! Cancel
SQL Connection Parameters
Parameter
Remote user na
Password
Server
Database
Connection List Report
OK
Cancel
4 of 4
F1 Help F7 Table Ctrl-PgUp Prev Ctrl-PgDn Next
CoEdit

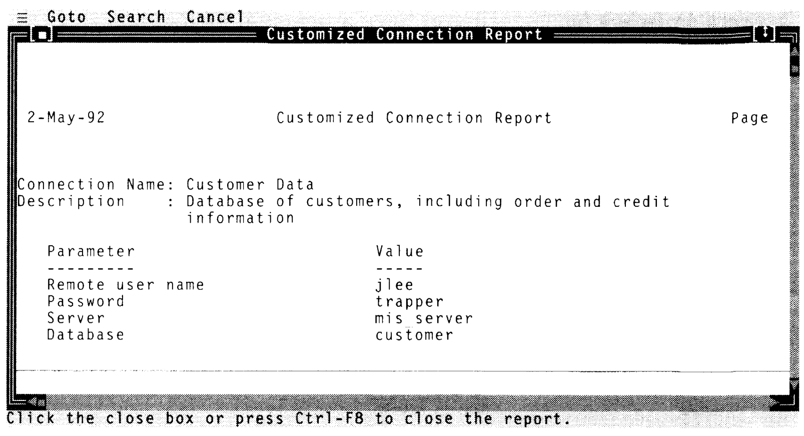
```
[X] Make Report:
    (•) To Screen
    ( ) To Printer
    ( ) To File
        conncust.rpt

[ ] Save Connection List As Table:
        conncust
```

Enter values you want to set as defaults for this connection, then press [F3] to save changes and return to the SQL Connection list.

The Connection List Report screen lets you create a report on the current session. You can display the report onscreen, print it, write it to a file, or create a Paradox table with the same data.

Printing reports

If you elect to create a report and send it to the screen, select the Make Report option and click To Screen, then choose OK. You'll see the "**Working...**" message, followed by a screen similar to this one:



```
2-May-92               Customized Connection Report              Page

Connection Name: Customer Data
Description    : Database of customers, including order and credit
                 information

   Parameter                         Value
   ---------                         -----
   Remote user name                  jlee
   Password                          trapper
   Server                            mis_server
   Database                          customer
```

Click the close box or press Ctrl-F8 to close the report.

Click the Close box or press *Ctrl-F8* to close the Report window and return to the summary screen.

To create a report and send it to the printer, select the Make Report option, click To Printer, then choose OK.

To create a report and write it to a file, select the Make Report option and click To File, type the file name, and choose OK. (If a report with the same name already exists, you'll be asked whether you want to replace the existing file.)

Saving a connection list as a table

To save the new connection list to a table, select the Save Connection List As Table option. Then type the table's name and choose OK. (If a table with the same name already exists, you will be asked whether you want to replace the existing table.) To view this table, exit Customize to the Paradox Main menu, choose View, and select the new table.

MakeReplicas

When you choose MakeReplicas, SQL Setup leads you through the steps of creating replicas. SQL Setup begins by displaying the Table Selection screen, which lets you select the criteria SQL Setup will use to retrieve tables from your database server. The Table Selection screen varies slightly for each server, but resembles the one shown in the following figure:

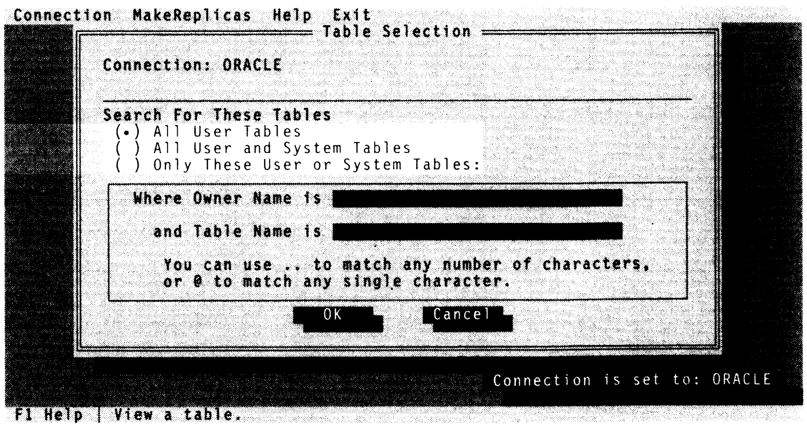

Connection MakeReplicas Help Exit

Table Selection

Connection: ORACLE

Search For These Tables
(•) All User Tables
( ) All User and System Tables
( ) Only These User or System Tables:

Where Owner Name is

and Table Name is

You can use .. to match any number of characters,
or @ to match any single character.

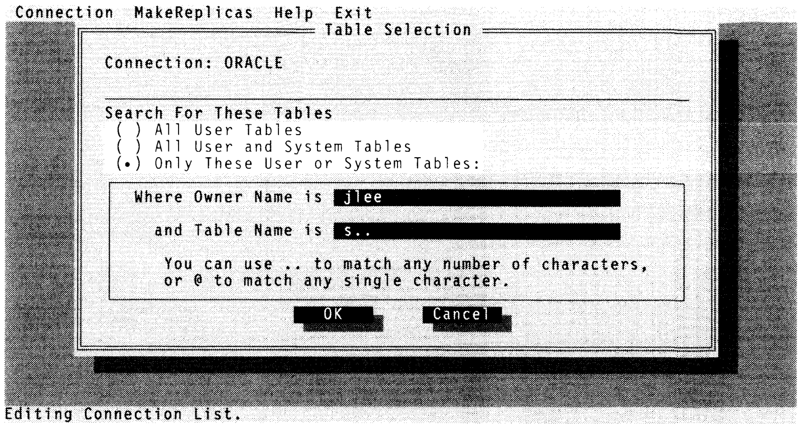
Searching for User and System tables

You can search for all user tables, all user and system tables, or specified tables. If you choose the All User and System Tables button, SQL Setup looks for all system tables, in addition to the user tables located on your server. For example, you might want to create replicas of system tables to use QBE to query catalog data on your database server.

You can also constrain the search by specifying the tables' owner or creator (if applicable for your database server), or enter characters common to the table names you want to select.

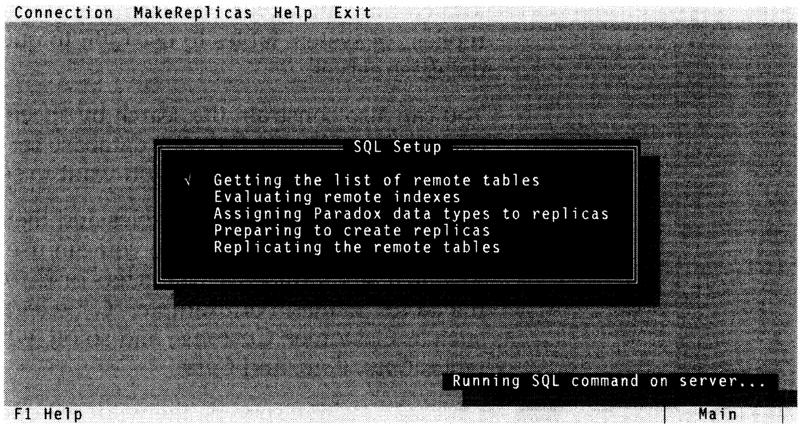
If you specify the names of tables and their owners using Only These User or System Tables option, you can use the wildcard characters .. to match any number of characters or @ to match a single character in the tables' names. For example, "**C..**" selects all tables starting with C, such as Customer, Customs, and so on, while "**C@PS**" only selects the tables *Caps*, *Cops*, and *Cups*.

Note On case-sensitive servers, the wildcard operators @ and .. always produce a case-insensitive search pattern.



```
Connection  MakeReplicas  Help  Exit
                      Table Selection

  Connection: ORACLE

  Search For These Tables
   ( ) All User Tables
   ( ) All User and System Tables
   (•) Only These User or System Tables:

      Where Owner Name is jlee

        and Table Name is s..

         You can use .. to match any number of characters,
         or @ to match any single character.

                       OK         Cancel

Editing Connection List.
```

Once you determine which tables to search for, choose OK. MakeReplicas displays a status screen with a flashing check mark (✓) opposite the current step.

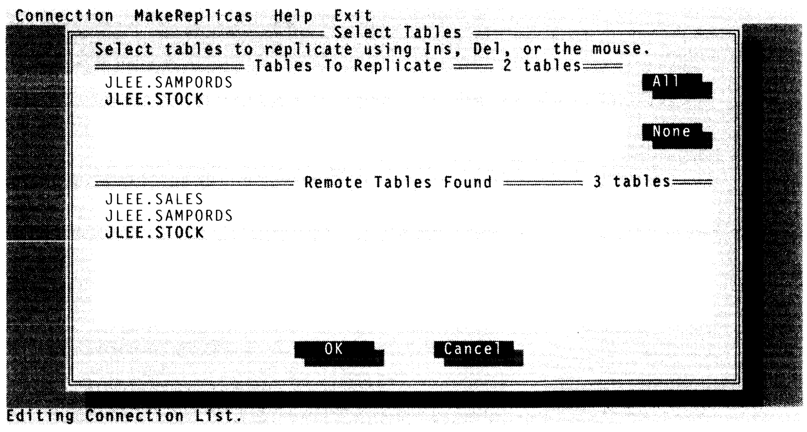


```
Connection  MakeReplicas  Help  Exit

                    SQL Setup

         √  Getting the list of remote tables
            Evaluating remote indexes
            Assigning Paradox data types to replicas
            Preparing to create replicas
            Replicating the remote tables

                                  Running SQL command on server...
F1 Help                                                    | Main |
```

SQL Setup will look in your server's system tables (or catalog) to obtain a list of tables that match the criteria you specified on the Table Selection screen.

Selecting tables to replicate

When SQL Setup has found all accessible tables that match the criteria you defined, it displays the following screen from which you choose the tables it should replicate.


```
Connection  MakeReplicas  Help  Exit
                      Select Tables
 Select tables to replicate using Ins, Del, or the mouse.
            Tables To Replicate  2 tables
 JLEE.SAMPORDS                                        All
 JLEE.STOCK

                                                      None

                 Remote Tables Found          3 tables
 JLEE.SALES
 JLEE.SAMPORDS
 JLEE.STOCK



                       OK        Cancel
Editing Connection List.
```

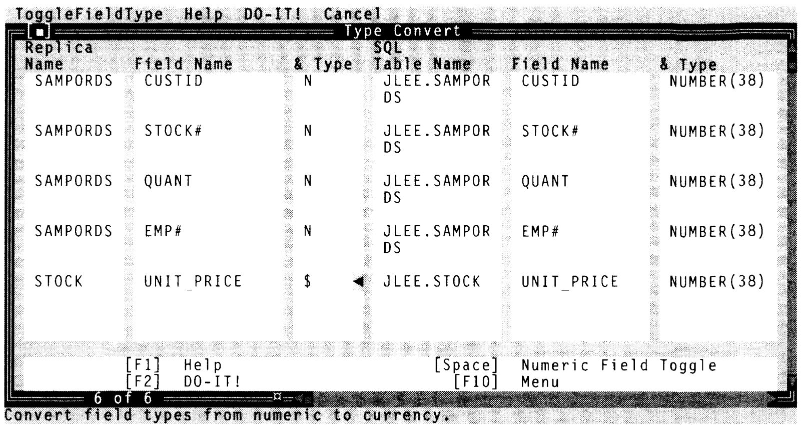
Use your mouse or arrow keys to select remote tables in the Remote Tables Found list box, then either double-click on the table name, or press *Ins* or *Spacebar* while the table name is selected to copy the table name into the Tables To Replicate list box. To remove a table from the Tables To Replicate list, you can double-click on a table name or press *Del* or *Spacebar* while the table name is selected.

To copy all tables listed in the Remote Tables Found list box to the Tables To Replicate list box, click the All button. To clear the contents of the Tables To Replicate list box, click the None button. If you want to replicate almost all of the tables in the Remote Tables Found list box, click the All button, then remove the tables you don't want to replicate from the Tables To Replicate list.

Converting numeric fields to currency fields

If any of the tables you selected in the Select Tables screen contain numeric fields, SQL Setup lets you determine on a column-by-column basis whether these field types should be interpreted as currency fields when viewed in Paradox. Even if your database server doesn't support currency field types (sometimes called MONEY fields), you can display numeric (N) fields as currency ($) fields in Paradox.

Note This conversion option appears only when you are replicating data from servers that don't support a currency field type. Otherwise, SQL Setup automatically maps your server's currency field type to a currency ($) field in Paradox.



```
ToggleFieldType  Help  DO-IT!  Cancel
                                Type Convert
Replica                                     SQL
Name       Field Name        & Type         Table Name    Field Name       & Type
SAMPORDS   CUSTID              N            JLEE.SAMPOR   CUSTID           NUMBER(38)
                                            DS
SAMPORDS   STOCK#              N            JLEE.SAMPOR   STOCK#           NUMBER(38)
                                            DS
SAMPORDS   QUANT               N            JLEE.SAMPOR   QUANT            NUMBER(38)
                                            DS
SAMPORDS   EMP#                N            JLEE.SAMPOR   EMP#             NUMBER(38)
                                            DS
STOCK      UNIT_PRICE          $    ◄       JLEE.STOCK    UNIT_PRICE       NUMBER(38)

           [F1]  Help                            [Space]  Numeric Field Toggle
           [F2]  DO-IT!                            [F10]  Menu
        6 of 6
Convert field types from numeric to currency.
```

Toggle between numeric and currency field types by pressing *Spacebar,* by selecting the menu choice, or by double-clicking on Field Type.

When you're finished specifying the fields to convert to currency, press *F2* Do_It! to save, or press *F10* Menu and choose Cancel to abandon your changes.

Naming replicas

SQL Setup automatically verifies that the new replica names do not conflict with the names of

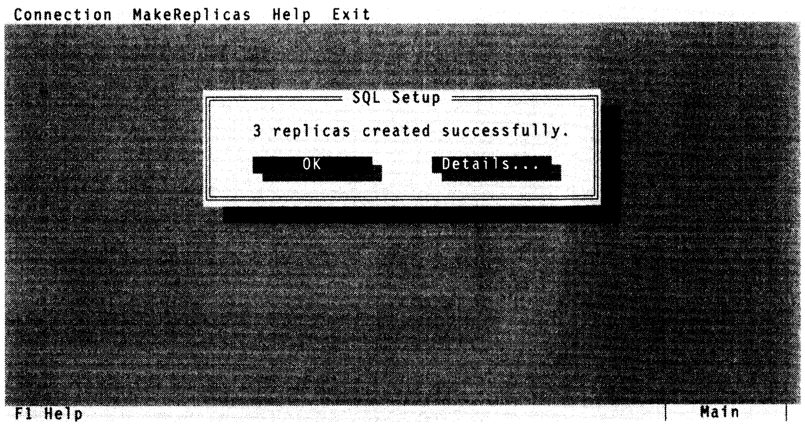
- any existing local tables
- any existing replicas that use the same connection and has an associated form or report
- any existing replicas that use a different connection
- any other tables in the list of tables to replicate

If a conflict arises, SQL Setup renames the conflicting table using *tablename-n*, where *n* is a number. For example, if your remote table is called *Customer* and SQL Setup detected a conflict with a local Paradox table with the same name, your replica would automatically be renamed *Custom-0.* This protects you from accidentally overwriting existing tables.

If SQL Setup resolves any table name ambiguities, it writes a list of all replicas it renamed to a table called *SQLprobs.* SQL Setup then displays the contents of *SQLprobs* to inform you of its actions. When you finish viewing this table, press any key or click the mouse.

You can rename, delete, or copy replicas created by SQL Setup using Tools | SQL | ReplicaTools. See Chapter 5 of this manual for details.

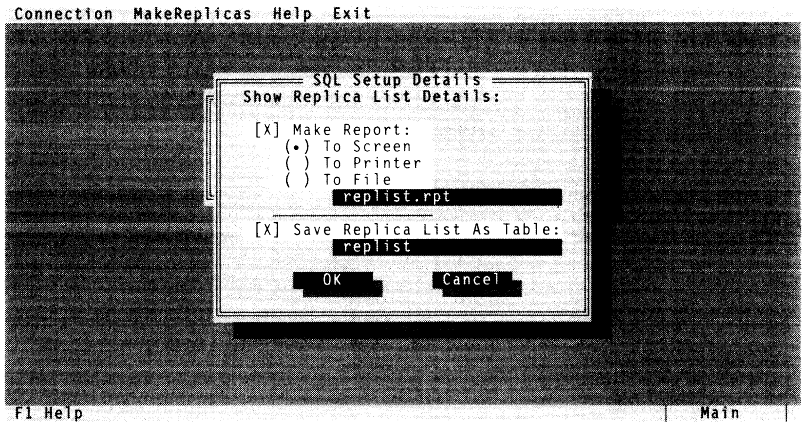
Once SQL Setup has created replicas for the remote tables, it displays a status message similar to the following:



```
3 replicas created successfully.
```

SQL Setup reports

To create a report on your session using SQL Setup, click the Details button shown in the previous figure.

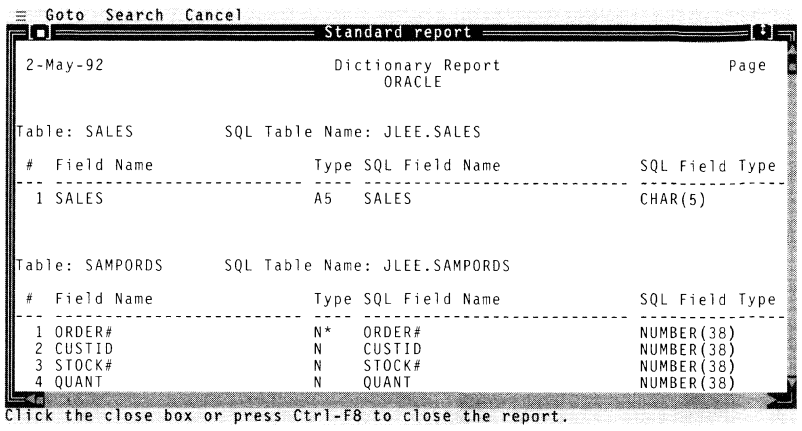


```
Show Replica List Details:

[X] Make Report:
    (•) To Screen
    ( ) To Printer
    ( ) To File
        replist.rpt

[X] Save Replica List As Table:
        replist
```

The SQL Setup Details screen lets you create a report on the current session. You can display the report onscreen, print it, write it to a file, or create a Paradox table with the same data.

Sending a report to the screen

If you elect to create a report and send it to the screen, select the Make Report option and click To Screen, then choose OK. You'll see the "**Working...**" message, followed by a screen similar to this one:



```
2-May-92                      Dictionary Report                      Page
                                    ORACLE

Table: SALES        SQL Table Name: JLEE.SALES

 #  Field Name                Type SQL Field Name              SQL Field Type
--- ------------------------- ---- --------------------------- --------------
 1  SALES                     A5   SALES                       CHAR(5)


Table: SAMPORDS     SQL Table Name: JLEE.SAMPORDS

 #  Field Name                Type SQL Field Name              SQL Field Type
--- ------------------------- ---- --------------------------- --------------
 1  ORDER#                    N*   ORDER#                      NUMBER(38)
 2  CUSTID                    N    CUSTID                      NUMBER(38)
 3  STOCK#                    N    STOCK#                      NUMBER(38)
 4  QUANT                     N    QUANT                       NUMBER(38)
Click the close box or press Ctrl-F8 to close the report.
```

Click the Close box or press *Ctrl-F8* to close the Report window and return to the summary screen. You can choose Details again to redisplay the report, or to print it.

Sending a report to the printer

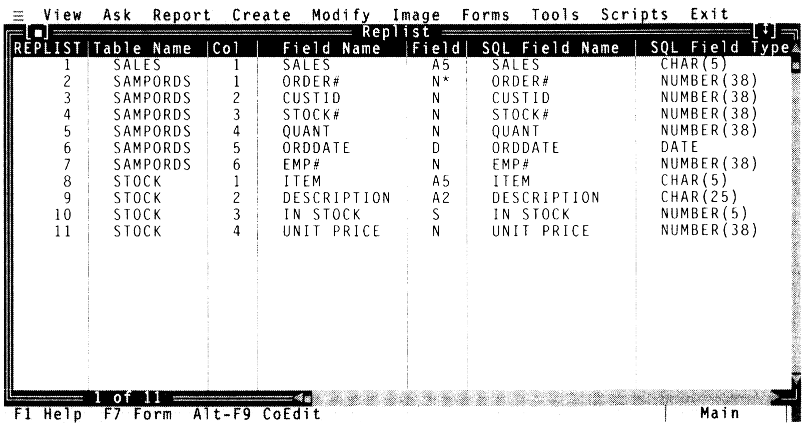
To create a report and send it to the printer, select the Make Report option, click To Printer, then choose OK.

Sending a report to a file

To create a report and write it to a file, select the Make Report option and click To File, type the file name, and choose OK. (If a report with the same name already exists, you'll be asked whether you want to replace the existing file.)

Saving a replica list as a table

To save the new replica list to a table, select the Save Replica List as Table option. Then, type the table's name and choose OK. (If a table with the same name already exists, you will be asked whether you want to replace the existing table.) To view this table, exit SQL Setup to the Paradox Main menu, choose View, and select the new table. The result resembles the following figure:



```
≡  View  Ask  Report  Create  Modify  Image  Forms  Tools  Scripts  Exit
                                  Replist
REPLIST|Table Name |Col |  Field Name  |Field| SQL Field Name | SQL Field Type
   1     SALES       1    SALES          A5    SALES            CHAR(5)
   2     SAMPORDS    1    ORDER#         N*    ORDER#           NUMBER(38)
   3     SAMPORDS    2    CUSTID         N     CUSTID           NUMBER(38)
   4     SAMPORDS    3    STOCK#         N     STOCK#           NUMBER(38)
   5     SAMPORDS    4    QUANT          N     QUANT            NUMBER(38)
   6     SAMPORDS    5    ORDDATE        D     ORDDATE          DATE
   7     SAMPORDS    6    EMP#           N     EMP#             NUMBER(38)
   8     STOCK       1    ITEM           A5    ITEM             CHAR(5)
   9     STOCK       2    DESCRIPTION    A2    DESCRIPTION      CHAR(25)
  10     STOCK       3    IN STOCK       S     IN STOCK         NUMBER(5)
  11     STOCK       4    UNIT PRICE     N     UNIT PRICE       NUMBER(38)

             1 of 11
F1 Help  F7 Form  Alt-F9 CoEdit                                    Main
```

Exit

When you choose Exit | Yes from the SQL Setup Main menu, you leave SQL Setup and return to the Paradox Main menu.

CHAPTER 8

The sample application

This chapter describes a few development scenarios and then presents the SQL sample application, a simple application that illustrates the use of SQL programming techniques in PAL.

PAL applications and the SQL environment

You can develop new PAL applications that use data from local tables, from remote tables, or from any combination of the two. Following are the basic steps in developing SQL Link PAL applications:

1. Run SQL Setup to access existing remote tables and customize your server connection (if necessary).
2. Load your application.
3. Select the server connection.
4. Run remote operations.
5. Commit or roll back your changes to end the transaction (if your application is transaction-based).
6. When finished, disconnect from the server.

Common approaches to building SQL Link applications

Following are a few different approaches you can take when designing and writing SQL Link applications. The first two scenarios use an airline ticketing application as an example, although many applications use the same transaction-processing techniques and concepts outlined here. The basic goal of these approaches is to lock records for the shortest time possible, while ensuring data integrity.

Sample approach A

This example and the next one are based on a customer trying to book a seat on a flight. The first approach sets AutoCommit to No at the last possible moment to achieve the highest level of concurrency. The drawback to this approach is that a plane seat could sell while the customer is making a decision.

1. The customer calls a reservations agent and asks which flights are available on a specific route.
2. The application turns AutoCommit on and queries for available flights (using Ask on a replica and filling in the query form).
3. The customer picks a flight.
4. The application turns AutoCommit off, and queries for the specific flight, placing a lock on the flight. If the seat is still available, the application performs a CHANGETO query to reserve the seat and then uses SQLCOMMIT to save the changes and release the locks.
5. The application turns AutoCommit on again.

Sample approach B

The second approach is similar except it sets AutoCommit to No at the first customer query. The drawback to this example is that all users pay for the increased control of one user and you don't maintain the highest level of concurrency. The advantage is that a seat cannot sell while the customer is deciding if they want it or not.

1. The customer calls a reservations agent and asks which flights are available on a specific route.
2. The application turns AutoCommit off and queries for available flights. This locks all of the flights returned by the query.
3. The customer picks a flight.
4. The application performs a CHANGETO query to reserve the seat and then uses SQLCOMMIT to save the changes and release the locks.
5. The application turns AutoCommit on again.

The SQL sample application

The SQL sample application is based on the "read-before-write" approach, which combines the best features of sample approach A and sample approach B. The read-before-write approach differs from sample approach A because it compares the existing data to the new data before making any changes; it differs from sample approach B because it sets AutoCommit to No at the last moment instead of locking earlier. The SQL sample application is based on a database of customer information and a related database of orders placed by those customers.

1. A customer calls to say that they have moved and would like to update their mailing address.
2. The application then turns AutoCommit on and queries for that customer's data. It then uses COPYTOARRAY to save the record from *Answer* until later.
3. The application uses Tools | Copy | JustFamily to copy all of the family members from the replica to the *Answer* table.
4. The application then uses PICKFORM to change to the correct form, goes to Edit mode (not CoEdit because *Answer* is private), sets IMAGERIGHTS UPDATE to preserve the key, and uses a WAIT RECORD to allow editing. It then uses COPYTOARRAY to save the user's changes in an array.
5. When the user is finished editing, the application turns AutoCommit off and reissues the original query. This locks the record. The application then compares *Answer* with the original version from step 2. If they are the same, the application uses a CHANGETO query to update the table with the user's changes, commits, and turns AutoCommit back on. If they are different, someone else has changed the data and the application alerts the user.

This approach is based on a business rule that assumes it's rare that the same customer record will be modified by two different people at the same time. It uses the approach of verifying that the record hasn't been changed by another user before making any changes, rather than locking the record when it is accessed. If the record has been changed, the application tells the user that the record has changed and lets them refresh the editing screen. This approach provides high concurrency and less chance for deadlocks because locks are held for the shortest time possible.

The SQL sample application shows practical techniques and approaches for writing a PAL SQL application, including how to

- combine PAL and SQL commands to access remote data in a simple but powerful way
- take advantage of the transaction processing features available in the SQL environment
- display multi-table views (one-to-many relationships)

- use Paradox's cross-tabulation, graphing, and reporting features on remote data
- employ Table Browse techniques to edit records in remote tables
- respond to errors on the remote server using PAL's error-handling features

The SQL sample application is a multi-user PAL application that accesses customer and order information stored on your database server using SQL commands and functions.

The SQL sample application consists of these scripts:

Instlsql	Installs the SQL sample application by copying the proper tables to the server and creating the corresponding replicas. If the user has not established a server connection, brings up the server connection screen.
Make	Plays each script to create the sample application library.
Startsql	Starts the SQL sample application.
View	Lets the user view remote customer records.
Modify	Lets the user add, delete, or edit user information in the remote *Customer* table.
Orders	Lets the user add, delete, or edit order information in the remote *Orders* table.
Reports	Creates reports for the SQL sample application.
Menus	Contains procedures for enabling and disabling menu items depending on the current application context.
Utils	Utility procedures for the application.
Sqlerror	Provides error handling for the application.

You can print out each of these scripts with Scripts | Editor | Edit *<Scriptname>*, *F10*, Print. In addition, the Sample Application Disk contains various objects (primary indexes, forms, validation tables, reports, and graphs) that are part of the SQL sample application. The SQL sample application uses the following tables:

Customers	Table created on server using data from the source *Custdata* table.
Order	Table created on server using data from the source *Orddata* table.
Stock	Lookup table created on server using data from the source *Stkdata* table.
Tcust, Tord	Local utility tables used for forms and reports.
Privcust, Privord	Created on application startup in the private directory.

Installation

You need to install the SQL sample application before you begin using it. Use the installation program that comes with SQL Link to install the sample application in a directory on a local or network hard disk. If you haven't already done this, refer to your server-specific addendum for instructions.

Next, start Paradox SQL Link and run the installation script called *Instlsql* to set up the SQL sample application environment. The *Instlsql* script creates the library for the sample application (SQLAPP.LIB) and creates the sample *Customer, Orders,* and *Stock* tables on your database server. You only need to run the *Instlsql* script the first time you run the sample application.

Note If a *Customer, Orders,* or *Stock* table already exists in your local Paradox environment, you will need to delete or temporarily rename the table before installing the sample application. If any of these tables exist on your remote server, *do not* drop them. To make your own copy of the sample application, create a sample database or user on your server and install the application there.

To run *Instlsql,*

1. Choose Tools | More | Directory from the Main menu.
2. Type the drive and path of the sample application directory, then press *Enter.* For example, if it resides in the C:\PDOX40\SQLSAMP directory, type

   c:\pdox40\sqlsamp

   and press *Enter.*
3. When Paradox asks you to confirm your choice, choose OK.
4. Choose Scripts | Play from the Main menu.
5. For the script name, type **instlsql** and press *Enter.*

6. The installation script asks you to select the server connection (if a current server connection isn't active) to use for the SQL sample application. Move the cursor to the server connection you want and press *F2* Do_It!, or press *Esc* to quit.
7. If you need to supply additional information for this server connection (such as user name or password), type in this information and press *F2* Do_It! to continue.

   If the information you specify is not correct, you return to the list of server connections. Repeat step 6.

The installation script creates the SQLAPP.LIB file and copies the *Custdata*, *Orddata*, and *Stkdata* tables to the database server, renaming these tables *Customer*, *Orders*, and *Stock*. When it copies these tables, Paradox SQL Link creates replicas (named *Customer*, *Orders*, and *Stock*) in the sample application directory.

Starting the sample application

The *Startsql* script starts the SQL sample application. It prompts you to select a server connection (if a current server connection isn't active) and calls the Main menu. *Startsql* is the only executable script in the sample application.

To start the SQL sample application,ParadoxSQL Link;sample application

1. Choose Tools | More | Directory from the Main menu.
2. Type the drive and path of the sample application directory, then press *Enter*. For example, if it resides in the C:\PDOX40\SQLSAMP directory, type

   c:\pdox40\sqlsamp

   and press *Enter*.
3. When Paradox asks you to confirm your choice, choose OK.
4. Choose Scripts | Play from the Main menu.
5. For the script name, type **startsql** and press *Enter*. The sample application uses the Paradox PRIVTABLES command to ensure that each user has their own local copy of *Tcust* and *Tord*. These tables contain the forms used by the application.
6. If you have not selected a server connection, the script prompts you to select the server connection you want to use for the SQL sample application. Move the cursor to the server connection you want to establish and press *F2* Do_It!.

7. If you need to supply additional information for this server connection (such as user name or password), type in this information and press *F2* Do_It! to continue.

   If the information you specify is not correct, you return to the list of server connections. Repeat step 6.

Startup note A private directory must be specified for your copy of Paradox before you attempt to use the sample application. The directory you designate cannot be the directory in which the sample application resides. You can use Tools | Net | SetPrivate to set or change a private directory during a single Paradox session. You can use the Custom Configuration Program to set a default private directory. For more information on designating private directories, see Chapter 17 of the Paradox *User's Guide*.

Multi-user access To allow other users (other than the owner/creator of the sample application tables) to run the SQL sample application, you need to grant privileges on the *Customer*, *Orders*, and *Stock* tables created during installation. The privileges you need to grant are listed in the following table:

Table	Granted privileges
Customers	SELECT, UPDATE, INSERT, DELETE
Orders	SELECT, UPDATE, INSERT, DELETE
Stock	SELECT

Note If you're unfamiliar with granting privileges, refer to your server documentation.

Once the installation is complete, the SQL sample application Main menu appears on the screen.

The *Startsql* script shows how the SQLCONNECTINFO function determines whether the user has selected a server connection and, if not, uses the SQLSELECTCONNECT command to present the user with a list of available server connections to choose from. The connection the user selects becomes the active connection for the SQL sample application.

The Main menu

The SQL sample application Main menu consists of four menu selections:

- System menu (≡) features a Connections submenu (with options for selecting a server) and an About command, which tells you about the SQL sample application.
- Table menu features selections for viewing and editing customer data.
- Report menu features selections for printing reports. Users can request reports on all records or West Coast records only. Other selections allow the user to view a graph of orders summarized by state, view a crosstab report of orders by product by state, and change the discount percentage applied to customer orders.
- Exit menu allows the user to close the SQL sample application by selecting Exit | Yes. The user can select Exit | No to resume work in the SQL sample application.

A closer look at the scripts used by the SQL sample application

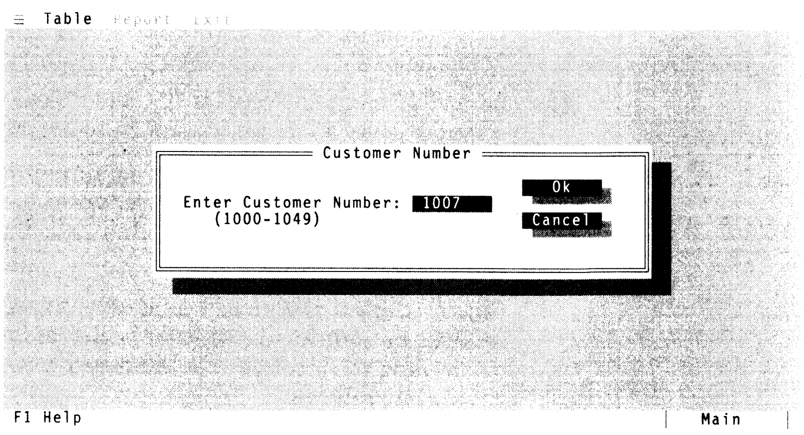
The following section provides some details and code examples from the SQL sample application. These examples illustrate how to view and update server data and how to query and create reports from server data. PAL programmers can also examine the complete script files (found in the sample application directory) for more details.

Error handling

Throughout the SQL sample application, errors trigger a PAL error-handling routine. Whenever an error occurs, a dialog box appears, which displays the local or server error. The *Sqlerror* script sets the global variable *ErrorFlag* to True and creates error message strings using Paradox SQL Link's error-handling functions (SQLERRORCODE() and SQLERRORMESSAGE()) and the Paradox error-handling functions (ERRORCODE(), and ERRORMESSAGE()). If *ErrorFlag* is True, the script rolls back the current transaction and displays the appropriate error messages.

Viewing customer information

Choose Table | View to view individual customer information records. The *Viewcust* script employs a dialog box where the user can enter a customer number. If the user enters a valid customer number, the corresponding information appears in a form (as in the following illustration). The user can open another customer selection dialog box with the *Ctrl-Z* key combination, or they can press any other key or click a mouse button to close the form.



The *Viewcust* script consists of these procedures:

- *getCustNoBox* prompts the user to input a customer number. The range of valid customer numbers is displayed in the dialog box. The user can press OK to use the customer number or Cancel to return to the main menu.
- *selectCust* runs a SQL NOFETCH query on the server, then uses SQLFETCH to transfer the customer information to an array. (Sets the cursor on the server to the position specified by customer number.)
- *viewCust* displays the selected information in a form, if the customer record is found on the server.

Adding, deleting, and editing customer information

Choose Table | Modify to add, delete, or edit customer records. The *Modify* script uses a local table view of the server data. The user can perform all modifications on the local data table, and changes are reflected on the server.

The script creates a local table for each user that accommodates one screenful (or page) of information. The *FillPage* procedure fetches records from the current cursor position on the server into the local table until the page is full or the last record on the server is reached.

The *insertCust* procedure handles the addition of new customer records to the server and checks for key violations. The procedure uses passthrough SQL statements to add the customer to the server. This technique is illustrated in the following code fragment:

```
SQL
  INSERT INTO ~FullCustomerName~ (Customer_No, Last_Name,First_Name, Address,
City,State, Zip_Code,Telephone, Discount)
  VALUES (~SQLVAL([Customer_No])~, ~SQLVAL([Last_Name])~,
     ~SQLVAL([First_Name])~, ~SQLVAL([Address])~,
     ~SQLVAL([City])~, ~SQLVAL([State])~,~SQLVAL([ZIP_CODE])~,
  ~SQLVAL([Telephone])~,
     ~SQLVAL(IIF(ISBLANK([Discount]),0,[Discount]))~)
ENDSQL
```

Note SQLVAL() is used to ensure that data types are handled correctly and proper values are placed in a SQL statement.

The *delCust* procedure requests confirmation from the user before deleting and verifies that the selected record has not been changed by another user. The *delCust* procedure calls the *readBeforeWrite* procedure to check the record.

The *readBeforeWrite* procedure is also called by *changeCust,* the record editing procedure. *readBeforeWrite* allows for maximum concurrency by locking the record at the last possible moment. The code for this procedure follows:

```
;-------------------------------------------------------------------------
; Procedure: readBeforeWrite
; Description:
;     Performs a read before write validation of data.
;     Pass the procedure:
;        SQLTbl - Server table to query (case-sensitive)
;        SQLLocateVal - String containing Search Parameters
;        beforeRec - An array containing values of the record we want
;                    to compare.
;-------------------------------------------------------------------------

PROC readBeforeWrite(SQLTbl, SQLLocateVal, beforeRec)
   PRIVATE editFlag

   editFlag = (SYSMODE() = "CoEdit") ; test whether we're in CoEdit mode.
                                     ; If we are, set a flag
   IF (editFlag)                     ; and get us out of CoEdit.
      THEN DO_It!
   ENDIF
   SQLCOMMIT                         ; issue a COMMIT to ensure that
                                     ; we get current version of records
   SQLAUTOCOMMIT YES                 ; set autocommit to YES
   SQLRELEASE                        ; release any query that may be pending

   IF NOT (RunFetch(SqlTbl,"*",SQLLocateVal,""))     ; get current remote image
      THEN                                           ; error occurred ?
         errorFlag=FALSE
         SQLRELEASE                  ; release the cursor on remote table
         IF (editFlag)
            THEN COEDITKEY
         ENDIF
         RETURN -1
   ENDIF

   SQLFETCH SQLRec                   ; fetch original record to an array

   IF NOT Retval Then                ; unable to fetch record, or record deleted
     msgInfo("Problem","Unable to fetch original record. "+
           "It may have been deleted by another user.")
     SQLRELEASE                      ; release the cursor on remote table
     IF (editFlag)
        THEN COEDITKEY               ; return to CoEdit mode
     ENDIF
     RETURN -1
   ENDIF
   SQLRELEASE                        ; release the cursor on remote table

   FOR i FROM 2 TO ARRAYSIZE(beforeRec)  ; compare "before" record to
                                         ; "after" record
      IF (beforeRec[i] <> SQLRec[i])     ; "before" record and "after" record
                                         ;  are different
         THEN
            IF (editFlag)
               THEN COEDITKEY
            ENDIF
            RETURN 0
      ENDIF
   ENDFOR

   IF (editFlag)
      THEN COEDITKEY
   ENDIF
   RETURN 1

ENDPROC
WRITELIB "SQLApp" readBeforeWrite
RELEASE PROCS readBeforeWrite
```

The *Modify* script contains the following procedures:

- *modifyCust* is the main procedure for modifying customer records on the server through a local table view.
- *fillPage* fetches data from the server to fill the current page of the form.
- *dispatchEvent* handles and processes WAIT events.
- *mapQuery* performs a QBE update query using CHANGETO.
- *readBeforeWrite* provides a high-concurrency locking technique to ensure data integrity on the server.
- *changeCust* tests each record using the read-before-write procedure. If the records have not been changed by another user, it commits the changes.
- *insertCust* adds new customers to the *Customer* server table and checks for key (unique index) violations.
- *delCust* deletes customers from the *Customer* server table and executes a read-before-write procedure to verify integrity before committing changes.

Closing the customer table

Choose Table | Close to close the customer information table on the local desktop. The closing procedure verifies that all your changes have been committed.

Entering orders

Choose Table | Orders to add or update customer order information. The *compareRecs* procedure in the *Orders* script handles each change to the order information as a single transaction (using

SQLAUTOCOMMIT, SQLSTARTTRANS, SQLCOMMIT, and SQLROLLBACK) to ensure data integrity.

The *Orders* script uses a remote lookup table procedure, *itemSQLPick*, to display a list of products, letting the user quickly and accurately pick an item to enter on an order form. (The item lookup appears when a user presses *F1* with the cursor in the Item field.) The script also checks that the user entered a value in the Quantity Ordered field, calculates the extended price, and checks for invalid information in the Product Code field.

The *Orders* script contains the following procedures:

- *ordEntry* enters new orders and reviews existing records.
- *processWaitEvents* handles and processes WAIT events.
- *fetchOrders* queries the remote *Orders* table for all orders for the current customer. The answers are saved in separate records ("before" and "after" snapshots) to be used later in a comparison.
- *postOrderChanges* commits order changes if the "before" and "after" records compare OK.
- *cleanUp* deletes temporary comparison tables and returns the user to the *Customer* table view.
- *compareRecs* compares records item-by-item to verify that no other user has changed the order records. If any two items in a record do not compare, *compareRecs* rolls back the changes.
- *resetOrd* restores the local table after a rollback or comparison failure.
- *calculateOrderTotal* adds the line items together to generate an invoice total.
- *itemSQLPick* displays a list of stock items for the user's selection.

Printing records

Choose Report | All to print all records to the screen. The procedure *printAllRecords* in the *Reports* script can be changed to print reports to the printer.

Printing the West Coast records

Choose Report | West Coast to print all records to the screen. The procedure *printAllRecords* in the *Reports* script can be modified to print reports based on other criteria.

Graphing orders

Choose Report | Order Summary to display a graph of average orders by state. The procedure *doOrderGraph* in the *Reports* script can be modified to display different graphs.

Creating a crosstab report

Choose Report | Order by Product to create a crosstab report of orders by state by product. The procedure *doOrderByProduct* in the *Reports* script can be modified to display other crosstab reports.

Changing the discount percentage

Choose Report | Change Discount % to globally change the customer discount percentage. The procedure *setCustDiscPct* in the *Reports* script can be changed to allow percentage changes of more than or less than ten percent.

PAL and SQL commands

For more information about the PAL commands and functions used in the SQL sample application, see Chapter 6. For more information about SQL commands in the SQL dialect of your particular database server, see your database server manuals.

Glossary

Access privileges	Determines the degree to which you can read or modify information on the server; usually assigned by the database administrator. General access levels include rights to select, insert, update, create, and delete.
Back end	Relative to Paradox, a database server and its associated concurrency and data integrity control.
Client	In client-server architecture, the client software (residing on user workstations) sends requests from the workstation to the server for processing and displays the results of those requests back to the workstation.
Client-server architecture	In this environment, database management tasks are divided between the client, or "front end" software, which resides on user workstations, and the server or "back end" software, which resides on a dedicated database server or file server. Data is accessed on the client, then it is processed, stored, and managed on the server.
Commit	In transaction processing, you or the application commits a transaction to save changes made to data on the database server. *See also* Rollback, Transaction, Transaction processing.
Concurrency	The attribute of relational databases that allows all users to work with consistent data. Until you issue an implicit or explicit COMMIT or ROLLBACK statement, data required for your transaction will not be altered by any other user. *See also* Commit, Rollback, Transaction, Transaction processing.
Connection	Information that Paradox uses to connect to a server (for example, user name, password, database name, server name, and so on). *See also* Replica connection.

Connection parameters	Variables such as user name, server name, and so on, that your server requires for access. See your server-specific addendum for more information on connection parameters.
Data integrity	The accuracy and reliability of data.
Database	A collection of tables organized to serve a specific purpose.
Database administrator	The person responsible for managing SQL databases, including user access rights to data. Also known as the system administrator.
Database server	SQL database servers directly manage the database, including such tasks as processing requests, storing data, managing concurrent data access, and providing data security and integrity. *See also* Client.
Deadlock	When two or more users hold locks on data required by each other's SQL transactions, a deadlock has occurred. Database servers detect deadlocks and, in most cases, cancel one of the participating user's queries.
Dialect	A product-specific implementation of the standard Structured Query Language (SQL). The exact syntax of SQL statements might vary among database server products. *See also* Structured Query Language.
Dictionary	A local Paradox table that contains the structural information of the remote tables found on the database server. SQL Setup uses the dictionary to create replicas. *See also* Replica, SQL Setup Program.
Field	A category of information (column) in a table; a collection of related fields makes up one record (row) in a table. *See also* Record, Table.
File server	Stores programs and data shared by multiple users on a local area network (LAN) and allows users to share peripherals.
Front end	In Paradox SQL Link, the interface used to create and issue queries against a database server.
Index	A file that sorts records in a table to speed up searches for information and, in some cases, to ensure that the table doesn't contain duplicate or blank records. *See also* Key field.
Key field	One or more fields in a table used to build an index. *See also* Index.
Local	An operation that occurs at the client workstation, or a Paradox file stored on a floppy disk, workstation hard disk, or a LAN file server. *See also* Remote.

Local area network	A system that links PCs and lets them share data and peripherals. *See also* File server.
Lock	To preserve the consistency of each user's view of a database's data, relational databases must lock data once it has been accessed. Locked data cannot be changed by other users until it is implicitly or explicitly unlocked by the user (or, in the case of a deadlock condition, by the database server itself). *See also* Concurrency.
Multiuser	A system that lets more than one user access the same data at the same time.
NULL	A SQL field value that means "value unknown."
Object	A component of a database such as a column, table, or index.
QBE	*See* Query by example.
Query	A specific request for data.
Query by example (QBE)	A request for data formulated by providing an example of the answer you are looking for.
Record	A group of fields in a table that contain related information; a row in a table. *See also* Field, Table.
Relational database management system	Often abbreviated RDBMS, a database system based on the mathematical theory of sets in which information is organized into tables, or relations.
Remote	An operation that occurs on the database server, or a file that resides on the SQL database server. *See also* Local.
Replica	A local file that tells Paradox where to locate a remote table and how to work with the data in it. A replica contains structural information about the remote table and information that Paradox uses to connect to the database server.
Replica connection	A server connection based on the connection data found in the replica of a remote table. *See also* Connection.
Rollback	The ability to remove an unsuccessful transaction after an error. *See also* Commit, Transaction, Transaction processing.
Server	*See* Database server and File server.

SQL	*See* Structured Query Language.
SQL Setup Program	A SQL Link program that lets you create replicas for existing remote tables and add custom server connections.
Structured Query Language (SQL)	SQL has been adopted by the American National Standards Institute (ANSI) as the standard language for relational database management systems (RDBMS). SQL includes programming commands for data definition, data manipulation, data retrieval, security, and transaction processing. *See also* Dialect.
System administrator	*See* Database administrator.
Table	A structure of rows (records) and columns (fields) that contains information. *See also* Field, Record.
Transaction	A process or logical unit of work comprised of one or more database operations. *See also* Commit, Rollback, Transaction processing.
Transaction processing	A method of processing database information that increases data integrity and improves performance. At the conclusion of a transaction, you determine whether each operation in the transaction completed successfully, then either commit (save) or roll back (abandon) changes made to the data. *See also* Commit, Rollback, Transaction.
UseSQL	The SQL command editor that lets you compose SQL statements and send them directly to your database server. This program can be run anywhere in Paradox (with the *UseSQL* script) and specifically in the SQL Setup Program (with the UseSQL command).
Workstation	For Paradox's purposes, a DOS client to a database server, which issues SQL queries against that database.

INDEX

PARADOX® SQL LINK

BORLAND

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan and United Kingdom ▪ Part #23MN-PQL01-40 ▪ BOR 3154