PARADOX SQL LINK

CONNECTING TO INTERBASE™

BORLAND

Paradox SQL Link

Version 4.0

Connecting to InterBase

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

Printed in the U.S.A.

10 9 8 7 6 5 4 3 2 R1

CONTENTS

TABLES and FIGURES

Tables

Figures

Connecting to InterBase

This addendum to the Paradox SQL Link *User's Guide* shows you how to use Paradox SQL Link to connect to InterBase 3.3. You should also read the Paradox SQL Link *User's Guide* in addition to this addendum.

Installation

This section describes the hardware and software required to run SQL Link and tells you how to install SQL Link on your hard disk or network.

This section assumes the following:

- Your DOS client has a supported TCP/IP network software package installed and configured.
- Your InterBase 3.3 server is already installed.
- Paradox version 4.0 is already installed on either the shared disk of your network file server or the local hard disk of a DOS client.
- You have sufficient access rights (also called privileges) to the Paradox system files directory to add new files.

See your Paradox documentation for information on installing and configuring Paradox.

System requirements

The following sections describe the hardware and software you need to run SQL Link. Figure 1 is a diagram of the SQL Link to InterBase system architecture.

Database server requirements

To run SQL Link, you need the following server product and its associated network software installed and running:

- InterBase 3.3 or higher.
- TCP/IP must be installed on the server. See your InterBase documentation for details.

See your InterBase manuals for information on specific requirements.

Client requirements

Table 1 lists the requirements for your DOS client (the computer running SQL Link), which must be 100% IBM compatible.

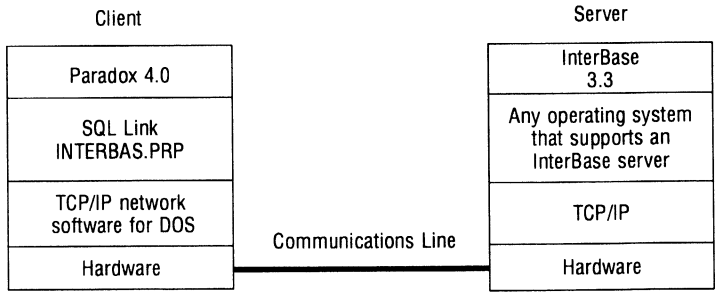
Table 1 Client requirements

Component	Requirement
Microprocessor	80286 or higher
RAM	A minimum of 2.5MB
DOS version	As required by your TCP/IP network software
Network software	Any supported TCP/IP network software package for DOS
Disk space	Approximately 1MB (for the SQL Link program itself)
Paradox	Version 4.0

Understanding the InterBase system architecture can help you identify common connection problems. All components of your system (see Figure 1) must be running before your DOS client can connect to a server. In addition, you *must* have a valid user name and password in the InterBase database *isc.gdb*.

Note Access privileges for InterBase clients are stored in the InterBase database *isc.gdb*. Database administrators can add or modify these privileges through **gsec**, an InterBase utility that is installed with InterBase 3.3. See your InterBase documentation for specific information on using **gsec**.

Figure 1 System architecture for InterBase



See your InterBase manuals for more information on specific requirements.

SQL Link must be installed in the Paradox system files directory (usually C:\PDOX40). This directory can be either on the local hard disk of a DOS client or on the shared disk of a network file server. Although you can name this directory differently, the SQL Link INSTALL program uses PDOX40 as the default. The Paradox driver for InterBase (INTERBAS.PRP) must be stored in this directory as well.

Installation steps

These installation instructions tell you how to install the SQL Link files, SQL Setup, and the SQL sample application in your Paradox system files directory.

If you have a PARADOX.DSQ file

If you have already defined a list of customized connections (a PARADOX.DSQ file), you must update the connection list each time you install a new SQL Link connection. Follow the instructions in Chapter 7 of the SQL Link *User's Guide* for adding a new connection.

The SQL Link Installation Disk contains an installation program, called INSTALL.EXE, that verifies your hardware setup and copies the SQL Link program files into your Paradox system files directory. The installation program also copies the InterBase message file INTERBAS.MSG and CONNECT.EXE (a utility that tests your InterBase connection from DOS) to your hard disk. Before you install SQL Link, make a backup copy of the SQL Link system disks, and keep the originals in a secure place.

To run INSTALL,

1. Insert the SQL Link Installation Disk into drive A. At the DOS prompt, type

   a:install

   and press *Enter*. (If you use drive B, type **b:install** and press *Enter*.) You'll see the introductory screen for installation.

2. Press *Enter* again to continue, and follow the prompts.

3. Choose the appropriate TCP/IP network software package (that is, the TCP/IP package you have installed on your DOS client) from the list of packages on the screen. INSTALL will copy the appropriate drivers and support files to the hard disk of your DOS client.

4. Choose the Location for InterBase Message File option if you want to install the InterBase message file in a directory other than C:\INTERBAS (the InterBase message file contains the text of InterBase error messages). If you choose to install this file in a different directory, you must set the DOS environment variable INTERBASE to the new location. For example, if you install the InterBase message file in C:\MESSAGE, you would add the following line to your AUTOEXEC.BAT file:

   SET INTERBASE=C:\MESSAGE

5. Choose the sample application option if you want to copy the SQL sample application to your hard disk. To install the SQL sample application on your server, run the *Sqlinst* script from within Paradox. For more information on the sample application, see Chapter 8 of the SQL Link *User's Guide*.

Note At any time during installation, you can press *Esc* to exit installation and return to DOS.

Information you need

Before starting SQL Link, make sure you have

- The information you need to log on to your server (user name, password, and database name).
- A working TCP/IP connection to your InterBase server.
- Set an INTERBASE environment variable if you installed the InterBase message file in a directory other than C:\INTERBAS.
- A HOSTS file that contains the name and IP address of each server that you plan to attach. At a minimum, the HOSTS file must contain the name and IP address of one host, like the following example:

  128.127.50.12 mis_server

Important

- A SERVICES file that contains the correct protocol for InterBase server access. At a minimum, the SERVICES file *must* contain the following line:

  gds_db 3050/tcp

Also, read the READSQL.IB file for additional information. To read this file, make the Paradox system files directory the current directory and type

readme readsql.ib

To print this file from the Paradox system files directory, type

print readsql.ib

Starting Paradox

To start SQL Link, make the Paradox system files directory the current directory, and start Paradox with the usual command:

paradox

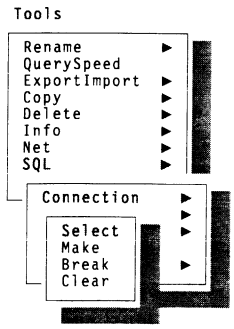
Without SQL

If you want to run Paradox without using its SQL capabilities, start Paradox with the following command:

paradox -sql off

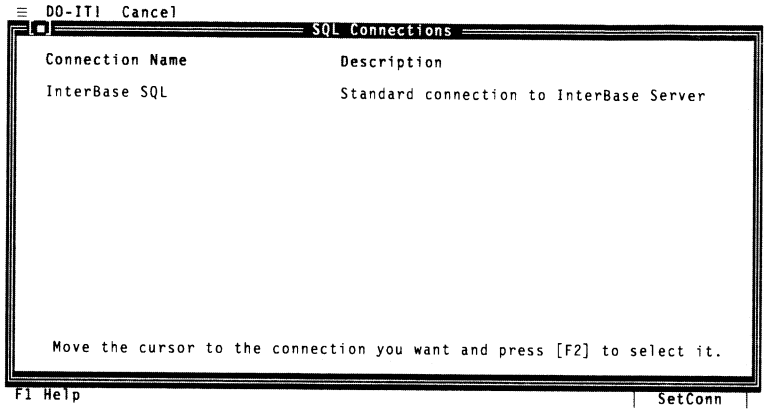
Selecting an InterBase connection

You must connect to your server before you can access a remote table. To select a server connection,



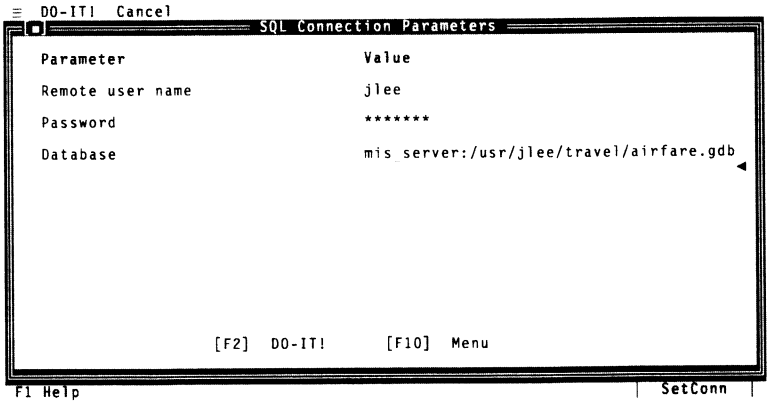
1. Choose Tools | SQL | Connection | Select from the Paradox Main menu.

   SQL Link displays a list of available server connections. Only those products that you install (or customize with SQL Setup) are displayed on this screen.



Note If you don't see a SQL choice on the Tools menu, you may have a problem with your SQL Link installation. Check the Paradox system files directory for PARADOX.OV3 and a file with a .PRP extension. If you don't see these files, reinstall SQL Link by following the installation steps described earlier in this addendum.

2. Select the server connection you want, then press *F2* Do_It!. Paradox displays the parameters for the InterBase connection.



3. Type the appropriate parameter values and press *F2* Do_It!. For information on the parameters, see the following section, "Connection parameters."

Connection parameters

The Paradox SQL Connection Parameters screen for the InterBase connection has three parameters, as shown in Table 2.

Table 2 Connection parameters

Parameter	Description	Example
Remote user name	Your login name on the host. The login name is assigned by the database administrator.	jlee
Password	Your password on the host. The password is assigned by the database administrator.	trapper
Database	InterBase database that you want to access. You can select any database to which you have been granted privileges on any platform supported by your hardware.	mis_server:/usr/interbase/examples/atlas.gdb

For more information about selecting server connections and providing the connection parameters your server requires, see Chapter 5 of the SQL Link *User's Guide*.

Troubleshooting common connection problems

If you run into problems when you first use SQL Link to establish an InterBase connection, try to isolate the problem using these steps:

1. Run the CONNECT utility to see if you can connect to the InterBase server from your DOS client. The CONNECT utility is a DOS program that attempts to connect to an InterBase database. The syntax is

   CONNECT <database name> <user name> [<password>]

   The following example shows you how to use the CONNECT utility:

   connect mis_server:/usr/interbase/examples/atlas.gdb jlee trapper

   If the CONNECT utility is successful, it will return the InterBase version ID of each link between your DOS client and the database. If CONNECT is successful and you are unable to use SQL Link to establish a connection to your server, you may have a problem with your SQL Link installation. Reinstall SQL Link by following the installation steps described earlier in this addendum.

   Also check the SERVICES file for the correct protocol for InterBase server access:

   gds_db 3050/tcp

   If the CONNECT utility is unsuccessful, use the following steps to test the lower-level protocols.

Note Troubleshooting steps 2 and 3 require a telnet program and a PING program. These DOS programs are available from your TCP/IP network software vendor and are *not* included in the SQL Link package (your TCP/IP network software package may use different names for these programs). If you don't have these programs on your DOS client, ask your network administrator to perform these tests for you.

2. Attempt to connect to the server with the telnet command to ensure that the TCP libraries are correctly installed (telnet is a remote virtual terminal emulator).

   If you are successful, the server will request your login name. You can check for the presence of the database you are trying to attach. If you receive a message like

   can't resolve hostname

   from your telnet program, check the HOSTS file to ensure that you have an entry for your host's name and IP address, like the following example:

   128.127.50.12 mis_server

   If telnet is successful and CONNECT is not, you may have a problem with your InterBase installation. See your database administrator (the person in charge of managing your database) for assistance.

3. PING the server to check that it is running and visible to your DOS client.

   If PING is successful but the telnet command is not, then there may be a problem with the inet daemon. If you cannot PING the server, you may have a routing problem.

Note If you don't have PING on your DOS client, you can PING the DOS client from the server node (if you have access to the server node). Ask your network administrator for instructions.

   If you are unable to PING the host from your DOS client, report the problem to your network administrator.

4. Go to the server node and attempt to attach the database. If you can successfully attach the database from the server node but not from your DOS client, you may not have a login set in *isc.gdb*. See your database administrator for assistance.

Access privileges

SQL Link fully supports database server security. In addition, Paradox passwords can be used to enhance server security by offering an additional layer of password protection. See the Paradox *User's Guide* for information on Paradox security.

You need a valid user name and password on your database server. Your database administrator can help you obtain these access privileges. You also need the appropriate privileges on the database you are trying to access.

Granting privileges

The SQL GRANT command is used to grant privileges. The syntax for the GRANT command differs depending on the class of privilege being granted. The system administrator or database administrator can grant access to SQL databases, programs, tables, and views.

Relations created through GDML

When you create a new relation in InterBase using GDML, *by default* (that is, you did not assign a GDML security class to the relation at create time), every other user on your server has all privileges to that relation. If you want to restrict access to the relation, you can use the SQL GRANT command to specify other users you want to have privileges on the relation.

Note

Once you use the SQL GRANT command on an InterBase relation, the relation is treated as if it was created through SQL commands. You *cannot* reverse the access restrictions placed on the table. You can, however, give all users access to the table with the SQL command

GRANT ALL PRIVILEGES
ON *<Tablename>*
TO PUBLIC

Tables created through SQL Link

When you create a new table through SQL Link, standard rules for SQL privileges apply to that table. Once you create a table using SQL commands, you are its owner. You will need to grant privileges to others if they are to use the table.

For performance reasons, InterBase caches privileges from the first access of a table for the remainder of your session. This means that when another user changes privileges on a table that you have accessed during this session, you will not see the results of those changes until you disconnect and reconnect; you cannot force InterBase to update your privileges by committing your changes.

See your server manuals for more information about SQL privileges and the GRANT command syntax.

Paradox SQL Link and InterBase views

A *view* of a SQL table is like a window that shows a specified portion of one or more server views or tables. You can access views just as you do tables by using the SQL Setup Program to create replicas for the views you want to access.

Views provide an additional level of table security and can also be used to hide complex data relationships. A view is simply a table to anyone who accesses it, yet that view could be a subset of columns from several tables.

Using views on wide tables

The maximum Paradox record size for keyed (that is, indexed) tables is 1350 bytes; for unkeyed (unindexed) tables it is 4000 bytes. To replicate tables with larger record lengths, create a view on a subset of the fields in these tables, then replicate the view.

InterBase data types

InterBase to Paradox

When you query remote tables, SQL Link maps InterBase data types to Paradox data types. Table 3 lists InterBase data types and their corresponding Paradox data types.

Table 3 InterBase to Paradox data type mapping

InterBase	Paradox
SHORT	S
LONG	N
FLOAT	N
DOUBLE	N
DATE	D (returns date portion only)
BLOB	A255*
CHAR(n)	A1–A255*
VARYING(n)	A1–A255*
ARRAY	A18 (BLANK)

* Data longer than 255 characters is truncated.

Paradox to InterBase

When you create or copy to a remote table, SQL Link maps Paradox data types to InterBase data types. Table 4 lists Paradox data types and their corresponding InterBase data types.

Table 4 Paradox to InterBase data type mapping

Paradox	InterBase
A1..A255	VARYING(1)..VARYING(255)
N	DOUBLE
D	DATE
S	SHORT
$	DOUBLE

SQL to InterBase

When you use passthrough SQL to create or alter a remote table, you must use SQL data types. Table 5 lists SQL data types and their corresponding InterBase data types.

Table 5 SQL to InterBase data type mapping

SQL	InterBase
SMALLINT	SHORT
INTEGER	LONG
DATE	DATE
CHAR(n)	CHAR(n)
VARCHAR(n)	VARYING(n)
DECIMAL	LONG
FLOAT	FLOAT
LONG FLOAT	DOUBLE
BLOB	BLOB

Note The InterBase data type ARRAY is not supported by SQL.

Support for Binary Large Object (BLOb) fields

SQL Link lets you retrieve BLObs from and add BLObs to InterBase tables. There are two modes of operation:

- Text mode, the default mode, gets BLObs from the InterBase database and displays them in Paradox A255 fields. Only the first 255 characters of BLOb data are displayed. You can't ADD or UPDATE BLObs in text mode.
- Binary mode lets you retrieve a BLOBID from the server. The BLOBID is returned into an A255 field as an 18-character string.

  You can write BLObs to an InterBase table in binary mode. When this mode is active, you can create a BLOb on the server. The server returns a BLOBID which you use to assign data to the BLOb field.

All of the functions discussed in this section must be nested in a SQL...ENDSQL statement. These functions are designed to be used from within a PAL script and are case-insensitive. Only one function (:BLOBMODE, :BLOBGET, or :BLOBPUT) can be included in each passthrough SQL statement. A description of each function follows.

Creating and altering tables that contain BLObs

The InterBase DSQL **CREATE TABLE** and **ALTER TABLE** commands support tables with BLOb columns. To create or alter a BLOb, use SQL passthrough or UseSQL. The following example shows how to create the *Houses* table, which is used in the :BLOBGET and :BLOBPUT examples.

```
SQL
   CREATE TABLE HOUSES
      (ADDRESS VARCHAR(200),
       OWNER VARCHAR(200),
       PHOTO BLOB)
ENDSQL

SQL
    CREATE UNIQUE INDEX HOUSEINDEX
       ON HOUSES (ADDRESS)
ENDSQL

SQL
    COMMIT
ENDSQL
```

There are additional options available for defining BLOb columns in a table. See your InterBase documentation for details.

:BLOBMODE

Selects BLOb compatibility mode as Text or Binary (the default is Text).

Syntax

:BLOBMODE { Text | Binary}

Description

When BLOBMODE is set to Text, BLOb data is returned in an A255 field in the *Answer* table as an ASCII string. You can't ADD or UPDATE BLOb fields when BLOBMODE is set to Text. If there are more than 255 bytes of data in the BLOb, the first 252 characters, followed by three raised dots (···), are displayed.

When BLOBMODE is set to Binary, BLOb data is returned as an encoded BLOBID of 18 characters (the first two characters are 0x, followed by a hexadecimal representation of the BLOBID). This BLOBID is used in subsequent BLOb functions. When BLOBMODE is set to Binary, you can perform INSERT or UPDATE operations on BLOb fields on your server.

Use

Set BLOBMODE to Binary if you want to INSERT or UPDATE records that contain BLOb data. Set BLOBMODE to Text if you simply need to view the data as a text-only entry of 255 or fewer characters.

Errors

If you don't specify the mode, :BLOBMODE returns ERRORCODE()=1000, **General SQL error**. The corresponding SQLERRORCODE() returns 0 if :BLOBMODE is set to Text, or 1 if it is set to Binary. You can use this value to determine the current setting of :BLOBMODE. The corresponding SQLERRORMESSAGE() returns **:BLOBMODE is set to Text** or **:BLOBMODE is set to Binary**, based on the current setting of :BLOBMODE.

If you specify the mode as anything other than Text or Binary, Paradox returns ERRORCODE()=33, **Invalid argument**.

Example

To set BLOBMODE to Binary, put the following statement in a PAL script:

```
SQL
   :BLOBMODE BINARY
ENDSQL
```

To set BLOBMODE to Text, put the following statement in a PAL script:

```
SQL
   :BLOBMODE TEXT
ENDSQL
```

:BLOBGET

Retrieves the BLOb data identified by *BLOBID* to the binary file *Filename*. This function is available only when BLOBMODE is set to Binary.

Syntax

:BLOBGET *BLOBID, Filename*

Description

BLOBID is the encoded 18-character value returned from a query when BLOBMODE is set to Binary.

Filename is the DOS file name of the target file. If you specify a file name that already exists, it is overwritten. You can specify any valid DOS path name as part of the *Filename* parameter. If a file extension is not specified, it is set to ".BLB"; if you specify *Filename* as "FILENAME." (with a trailing period), *Filename* is created with no extension.

Errors

If *BLOBID* is no longer valid (for example, if another user has deleted or changed the BLOb you are trying to retrieve), Paradox returns ERRORCODE()=1012, **Object does not exist**.

If the specified file cannot be created, Paradox returns ERRORCODE()=1010, **General create error**. If you attempt to use this function when BLOBMODE is set to Text, Paradox returns ERRORCODE()=26, **Invalid PAL context**.

Example

```
SQL
   :BLOBMODE BINARY
ENDSQL

SQL
   SELECT OWNER, PHOTO FROM HOUSES
   WHERE ADDRESS="624 Prospect Street"
ENDSQL

VIEW "Answer"

SQL
   :BLOBGET ~[PHOTO]~, C:\\TEMP\\HOUSFOTO.PNT
ENDSQL

IF ERRORCODE() <> 0
   THEN QUIT ERRORCODE()
   ELSE RUN BIG "PAINTER C:\\TEMP\\HOUSFOTO.PNT"
ENDIF
```

In this example, the SQL passthrough statement

```
SQL
   :BLOBGET ~[PHOTO]~, C:\\TEMP\\TEMPFOTO.PNT
ENDSQL
```

could also be written like this:

```
PHOTOFILE="C:\\TEMP\\TEMPFOTO.PNT"
SQL
   :BLOBGET ~[PHOTO]~, ~PHOTOFILE~
ENDSQL
```

:BLOBPUT

Creates a BLOb on the server from data in *Filename* and returns the *BLOBID* in a single-row, single-column *Answer* table. This function is available only when BLOBMODE is set to Binary.

Syntax

:BLOBPUT *Filename*

Description

Filename is the DOS file name containing the data you want to assign to a BLOb field on the server. The DOS file *Filename* is accessed in binary mode (that is, *Filename* is read in as a stream of bytes).

Upon successful completion, this function returns a single-row, single-column *Answer* table with a temporary BLOBID to be used in ADD or UPDATE operations.

Errors

If the file doesn't exist, Paradox returns ERRORCODE()=1012, **Object does not exist**. If you attempt to use this function when BLOBMODE is set to Text, Paradox returns ERRORCODE()=26, **Invalid PAL context**.

Example

```
RUN BIG "PAINTER C:\\TEMP\\TEMPFOTO.PNT" ; create or edit a paint file
                                         ; with your favorite paint program

SQL
   :BLOBMODE BINARY
ENDSQL

SQL
   :BLOBPUT C:\\TEMP\\TEMPFOTO.PNT
ENDSQL                                   ; send the file to the server
                                         ; and get a new BLOBID
```

```
IF ERRORCODE() <> 0
   THEN QUIT ERRORCODE()
   ELSE
                                        ; The field [BLOBID] in the Answer
                                        ; table contains a temporary BLOBID
   VIEW "Answer"
   MENU {MODIFY} {DATAENTRY} {HOUSES}
      [PHOTO] = [ANSWER->BLOBID]        ; Assigns [BLOBID] from
      [ADDRESS] = "624 Prospect Street" ; the Answer table into the field
      [OWNER] = "J. Lee"                ; PHOTO on the server
   DO_IT!
ENDIF
```

Note Upon completion of the ADD operation, a new, permanent BLOBID is assigned to the newly created entry. If you query the table after assigning the BLOb, the resulting *Answer* table will contain a permanent BLOBID.

You can also ADD multiple BLObs to an InterBase table, as shown in the following code fragment:

```
n = 10
ARRAY blobid[n]
ARRAY address[n]
; load the address array here!

SQL
   :BLOBMODE BINARY
ENDSQL

FOR i FROM 1 TO n
   SQL
      :BLOBPUT ~"File"+strval(i)+".pnt"~
   ENDSQL
   VIEW "Answer"
   blobid[i] = [Answer->blobid]
ENDFOR

MENU {MODIFY} {DATAENTRY} {HOUSES}
FOR i FROM 1 TO n
   [photo] = blobid[i]
   [address] = address[i]
   DOWN
ENDFOR
DO_IT!
```

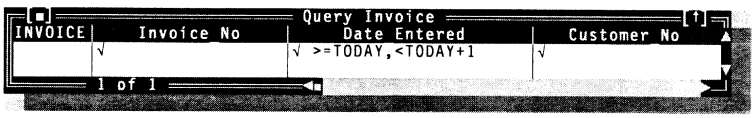
Query restrictions

Be aware of the following restrictions when querying remote tables:

- Pattern matching is always case-sensitive. For example, if your query contains the pattern **A..**, **Aurora** and **Atherton** would be found but **apple** would not.
- InterBase DATE columns can contain time information along with the date. When SQL Link maps this data to Paradox type **D**, the time portion of the DATE column is not read into Paradox.

  This can present a problem when you query a relation that contains DATE columns that have a time specified. For example, if you want to find all of the rows that were added to a relation today, you could use the following query:

  If any row contains a DATE column with a time specified, however, the query will not return those rows. Use the following query instead:



  This query will return all rows with today's date in the Date_Entered column that have no time specified. It will also return all rows with today's calendar date that contain times.

Index definition rules

The following rules apply to indexes on InterBase tables:

- A single-column index cannot exceed 252 characters.
- A compound index (multi-field key) cannot exceed 200 characters. This means that the total expression size (the sum of all columns that form the compound index) must be less than or equal to 200 characters. Furthermore, each indexed column's width is rounded up to the nearest multiple of four. For example, a column designated as **A49*** adds 52 characters to the total size of the index.

See the SQL Link *User's Guide* for more information about creating Paradox-compliant indexes. See your InterBase documentation for more information about InterBase data definition rules.

Transaction management in PAL programs

Paradox method of record locking

Paradox manages concurrent user access by locking the record the user is currently editing. If other users attempt to access the record, they will receive the message

Record has been locked by JLEE.

PAL programmers typically place explicit locks at the start of each edit operation, but this practice doesn't work as expected on SQL database servers. Placing row-level locks at the start of an operation fails to provide notification of lock conflicts. Users of these applications don't get the "**Record has been locked by...**" message; they must wait until an operation is completed and the record is available again.

Although a row-level locking strategy prevents user A from corrupting user B's data, it also has the effect of preventing the completion of aggregate operations such as SUM() or MAX(). The balance between optimal concurrent access and maximum data integrity protection can be effectively managed through careful programming that takes advantage of the server's strengths.

Database access using the read-before-write approach

Chapter 8 of the SQL Link *User's Guide* describes an approach to application development that applies to most SQL database servers. This "read-before-write" approach (also known as optimistic concurrency) is based on the assumption that it's rare that the same record will be modified by more than one user at any given time.

Although this approach works well on all SQL database servers, it doesn't take advantage of the InterBase multi-generational read technology. Multi-generational reads allow users to *automatically* receive the same benefits as the read-before-write approach. The application developer is able to write programs to access InterBase data without designing procedures to explicitly COMMIT transactions, lock rows, compare "snapshots" of data, and COMMIT once again.

InterBase transaction modes

There are two transaction modes supported by InterBase. Concurrency mode, the default, is used by all SQL client applications (including Paradox SQL Link) to access InterBase data. Concurrency mode supports multi-generational reads, which provide a stable view of the database. Concurrency mode doesn't support explicit row or table locks and, consequently, there is a greater risk of update conflicts (conflicts resulting from an attempt by one user to modify a

record that has already been modified by another user). The server automatically detects these update conflicts and issues an error.

Consistency mode, the alternate, is unavailable through SQL clients (including Paradox SQL Link). Although consistency mode transactions eliminate the risk of update conflicts, they do so at the expense of optimal multiuser concurrent access. If another InterBase user opens a relation in consistency mode, SQL Link clients will not be able to write to that relation.

For more about the differences between concurrency mode and consistency mode, see the discussion of transaction processing in the InterBase *Programmer's Guide*.

Writing PAL applications to access InterBase data

PAL programmers can design applications that leverage the strengths of InterBase 3.3 and provide a high degree of multiuser concurrent access while ensuring data integrity. The following examples illustrate two approaches to PAL application development.

Optimizing multiuser concurrent access

The first example shows an application that allows a higher degree of multiuser concurrent access. The drawback to this approach is that the likelihood of update conflicts is greater than that of the second approach.

This approach behaves like the read-before-write approach described in Chapter 8 of the SQL Link *User's Guide*. The main difference between this approach and the read-before-write approach is that this approach holds the transaction open while the user is editing the record. This approach also lets InterBase inform the program of any changes in the record on the server. This differs from the read-before-write approach described in the *User's Guide*, which relies on selecting the record again and comparing it to a "snapshot" of the original record to ensure that it hasn't changed.

The complete listing of a script that demonstrates optimized multiuser concurrent access follows. This script, MULTUSE1.SC, was copied to the SQLSAMP directory if you chose to install the SQL sample application.

```
; This code demonstrates how to optimize PAL applications for
; multiuser concurrent access.
;
; These examples use the CUSTOMERS table in the emp database, which is
; shipped with InterBase and gets installed as
; .../interbase/examples/emp.gdb. The replica is named with the first 8
; characters of the name, so the replica is called CUSTOMER.
; To create this replica, use SQL Setup

PROC myErrProc()
   PRIVATE ERRORPROC
   errorProcCalled = TRUE
   IF ERRORCODE() >= 1000                    ; if it's a SQL error
      THEN RETURN 1                          ; pass it back to the application
```

```
      ELSE RETURN 2                          ; call the debugger
   ENDIF
ENDPROC

PROC SelectRecordFromCustomers(custNo)
   QUERY

      Customer |   CUSTNO          |  CUSTOMER  |  CONTACT   |  ADDRESS   |
               | CheckPlus ~custNo | CheckPlus  | CheckPlus  | CheckPlus  |

      Customer |   CITY     |   STATE    |  ZIP_CODE |  PHONE_NO |  ON_HOLD  |
               | CheckPlus  | CheckPlus  | CheckPlus | CheckPlus | CheckPlus |

   ENDQUERY
   DO_IT!
   MOVETO 1
   CLEARIMAGE
ENDPROC

PROC EditAnswerTable()
   ; The following is a simple method for editing the answer table.
   ; You can use dialog boxes and other new UI features.
   ; In addition, you should preserve the key (CUSTNO) so that the user
   ; can't edit it.
   ;
   ; This procedure returns TRUE if user wants to post changes.

   MOVETO "Answer"
   COEDITKEY
   WINDOW MOVE GETWINDOW() TO -100, -100    ; move table view offscreen
   FORMKEY
   PROMPT "Press Esc to cancel, or F2 to save changes."
   WAIT RECORD
      UNTIL "F2", "Esc"
   DO_IT!
   RETURN RETVAL="F2"
ENDPROC

PROC WaitForEvent(theMsg)
   PROMPT theMsg                            ; set up prompt
   ECHO NORMAL                              ; show the prompt
   ECHO OFF                                 ; and turn echo off
   MESSAGE SQLERRORMESSAGE()
   GETEVENT MOUSE "down" KEY "all" TO x
   PROMPT                                   ; and clear the prompt line
   MESSAGE ""                               ; and the message
ENDPROC


; Start of program
MESSAGE "Reading record with custno = 102..."
ERRORPROC = "myErrProc"
errorProcCalled = FALSE
SQLAUTOCOMMIT NO
SQLROLLBACK

SelectRecordFromCustomers(102)              ; select the record for editing
MESSAGE ""                                  ; put the message away

done = FALSE
WHILE NOT done
   IF EditAnswerTable()                     ; allow the user to edit record
      THEN                                  ; returns TRUE if post changes
         ECHO NORMAL
         ADD "Answer" "Customer"            ; update the server with changes
         ECHO OFF
```

```
            IF errorProcCalled                ; the server returns an error
               THEN
                  errorProcCalled = FALSE     ; reset the flag
                  IF SQLERRORCODE() = "-912"
                     THEN                     ; update conflict
                        WaitForEvent("Press a key or click the mouse "+
                                     "to get the new record.")
                        SQLCOMMIT             ; autostart a new transaction
                        SelectRecordFromCustomers(102)  ; reselect the record
                     ELSE
                        QUIT SQLERRORMESSAGE() ; don't handle the error
                  ENDIF
               ELSE
                  done = TRUE                 ; no error occured
            ENDIF
         ELSE
            done = TRUE                       ; user wants to exit
      ENDIF
   ENDWHILE
   CLEARALL
   SQLCOMMIT
   ; End of program
```

Minimizing update conflicts

The second example shows an application that reduces the risk of update conflicts. The drawback to this approach is that only one user can access any row at a given time.

This approach simulates row-level locking to reduce the frequency of update conflicts. InterBase doesn't provide a row-level locking feature "on demand" and InterBase consistency mode is not available to SQL client applications (including Paradox SQL Link). This approach provides a level of deadlock protection beyond that of the first approach.

A user who starts a transaction requests an exclusive lock on a particular row. If no other user has a lock on that row, the user acquires the lock and reads from the row, and can then edit the row and post or abandon changes. Otherwise, the user must wait to acquire an exclusive lock on that row.

This example allows a user to lock a row by performing a "null" update on the row (a "null" update is an update that doesn't change any data). This ensures that other users must wait for this transaction to finish before they can read information from that row if they also use this approach; other users cannot update that row even if they don't use this approach.

Note When a user has an exclusive lock on a row, any other user who requests a lock on that row waits until the first user completes the transaction. This means that users must complete their transactions in a timely manner to ensure that others are not prevented from modifying the data for long periods of time.

Following is an example script that minimizes the risk of update conflicts. This script, MULTUSE2.SC, was copied to the SQLSAMP directory if you chose to install the SQL sample application.

```
; This code example shows an approach to application development that
; reduces the possibility of update conflicts.
;
; This example also uses the CUSTOMERS table in the emp database.
; myErrorProc, SelectRecordFromCustomers, EditAnswerTable
; are the same as in the previous model...
;
PROC myErrProc()
   PRIVATE ERRORPROC
   errorProcCalled = TRUE
   IF ERRORCODE() >= 1000                    ; if it's a SQL error
      THEN RETURN 1                          ; pass it back to the application
      ELSE RETURN 2                          ; call the debugger
   ENDIF
ENDPROC

PROC SelectRecordFromCustomers(custNo)
   QUERY

      Customer |   CUSTNO          |  CUSTOMER  |  CONTACT   |  ADDRESS   |
               | CheckPlus ~custNo | CheckPlus  | CheckPlus  | CheckPlus  |

      Customer |   CITY     |   STATE    | ZIP_CODE |  PHONE_NO | ON_HOLD  |
               | CheckPlus  | CheckPlus  | CheckPlus | CheckPlus | CheckPlus |

   ENDQUERY
   DO_IT!
   MOVETO 1
   CLEARIMAGE
ENDPROC

PROC EditAnswerTable()
   ; The following is a simple method for editing the answer table.
   ; You can use dialog boxes and other new UI features.
   ; In addition, you should preserve the key (CUSTNO) so that the user
   ; can't edit it.
   ;
   ; This procedure returns TRUE if the user wants to post changes.
   MOVETO "Answer"
   COEDITKEY
   WINDOW MOVE GETWINDOW() TO -100, -100    ; move table view offscreen
   FORMKEY
   PROMPT "Press Esc to cancel, or F2 to save changes."
   WAIT RECORD
      UNTIL "F2", "Esc"
   DO_IT!
   RETURN RETVAL="F2"
ENDPROC

PROC DoNullUpdateOfCustomers(custNo)
   PRIVATE lockedRecord
   lockedRecord = FALSE
   WHILE NOT lockedRecord
      QUERY

         Customer | CUSTNO  |      CUSTOMER          |
                  | ~custNo | _cust, changeto _cust  |
```

```
      ENDQUERY
      DO_IT!
      IF errorProcCalled                    ; the errorproc was called
         THEN
            errorProcCalled = FALSE         ; reset the flag
            IF SQLERRORCODE() = "-912"      ; update conflict
               THEN SQLCOMMIT               ; end transaction, start new one
               ELSE QUIT SQLERRORMESSAGE()  ; don't handle the error
            ENDIF
         ELSE lockedRecord = TRUE           ; no error occurred
      ENDIF
   ENDWHILE
ENDPROC

; Start of program
ERRORPROC = "myErrProc"
errorProcCalled = FALSE
SQLAUTOCOMMIT NO
SQLROLLBACK

MESSAGE "Waiting for lock on record with custno = 102..."
DoNullUpdateOfCustomers(102)

MESSAGE "Reading record with custno = 102..."
SelectRecordFromCustomers(102)              ; select the record for editing
MESSAGE ""

IF EditAnswerTable()                        ; allow the user to edit record
   THEN                                     ; returns TRUE if post changes
      ECHO NORMAL
      ADD "Answer" "Customer"               ; update the server with changes
      ECHO OFF
      IF errorProcCalled                    ; the server returns an error
         THEN
            SQLROLLBACK                     ; release the lock
            IF SQLERRORCODE()="-912"        ; update conflict
               THEN QUIT "We got an update conflict even though we had the "
                         "record locked. This should never happen."
               ELSE QUIT SQLERRORMESSAGE()  ; don't handle the error
            ENDIF
      ENDIF
      SQLCOMMIT                             ; save changes and release the lock
   ELSE SQLROLLBACK                         ; release the lock
ENDIF
CLEARALL
; End of program
```

InterBase system relations

InterBase includes a special set of tables called *system relations*, which provide information about production tables. You can access the system relations from SQL Link with SQL...ENDSQL or UseSQL, the SQL command editor. You can also create replicas for system relations with SQL Setup and then query them in SQL Link just as you would any other table. System relations describe privileges, indexes, remote table structures, and other metadata.

To help you troubleshoot problems, Table 6 lists a few InterBase system relations you might want to access through SQL Link.

Table 6 Selected InterBase system relations

Table name	Use
RDB$RELATIONS	Lists all tables and views
RDB$RELATION_FIELDS	Lists columns of tables and views
RDB$INDICES	Lists indexes

Additional reference tables for InterBase 3.3

Table 7 lists general items that you might find helpful in working with InterBase.

Table 7 General information

Item	Description
Does the server require an explicit SQLSTARTTRANS for multistatement transaction processing?	No
Product name	INTERBAS
SQL dialect	INTRBASE
Case-sensitive for data?	Yes (including patterns)
Case-sensitive for objects?	No

Table 8 lists field-naming rules.

Table 8 Field-naming rules

Naming rule	Paradox	InterBase
Max length (characters)	25	31
Valid characters*	All except " , [,], {, }, (,), –, >	Letters, digits, $, or _
Must begin with	Any valid character	Letters only (A–Z, a–z)

* Paradox field names cannot be the symbol # alone.

Note You cannot use reserved words as object names. See the *PAL Reference* and your server manuals for a list of reserved words.

PARADOX SQL LINK

BORLAND

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-8400. 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-PQI01-40 ▪ BOR 4701