Upsizing Your dBASE Application

This standalone Help system is designed for dBASE for Windows developers who have already installed one or more Borland SQL Link drivers and want to build scalability into their applications. To get information on a topic, click the underlined text.

Working With SQL Link Drivers - describes how the Borland SQL Link drivers provide access to SQL data through dBASE for Windows

SQL-enabled dBASE - references dBASE language extensions for working with SQL data, and other dBASE language elements that support SQL

Creating dBASE Applications for SQL Data - discusses strategies for dBASE for Windows developers who want to migrate to the SQL environment.

Working With SQL Link Drivers

See Also

Expand list

Topics

Essentials

Table Requirements

SQL Link Locking and Transaction Support

Working With SQL Link Drivers

See Also
Collapse list

Topics

See Also

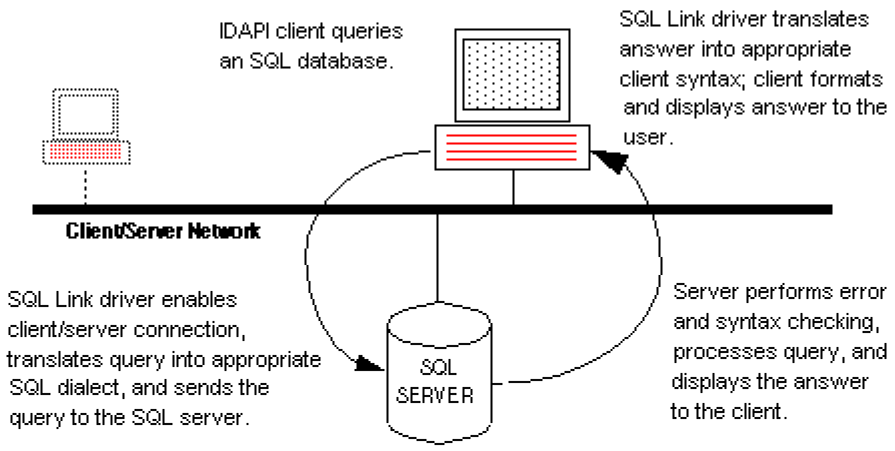
SQL-enabled dBASE

Creating dBASE Applications for SQL Data

Essentials

See Also

Borland® SQL Links for Windows is a collection of drivers that let you work with SQL data through supported Borland applications. The SQL Link driver enables the connection to the SQL server, translates queries into the appropriate SQL dialect, and passes them to the SQL database. When processing is complete, the SQL database returns the answer to the client in a format that the desktop application can display.


IDAPI client queries an SQL database.
SQL Link driver translates answer into appropriate client syntax; client formats and displays answer to the user.
Client/Server Network
SQL Link driver enables client/server connection, translates query into appropriate SQL dialect, and sends the query to the SQL server.
SQL SERVER
Server performs error and syntax checking, processes query, and displays the answer to the client.

Note: Borland database applications also support the use of SQL statements against local (Paradox or dBASE) data. For information on how to use Local SQL with dBASE for Windows, see the dBASE for Windows online Help.

Using dBASE with an SQL Link driver actually extends what you can do with your application, lending it some of the advantages inherent in using SQL databases:

- The ability to query the SQL server directly
- Support for SQL-style transactions
- Improved record caching
- Data locking behavior

Accessing an SQL server through an SQL Link driver and dBASE also lends traditional SQL database users some of the advantages inherent in workstation databases, enabling you to

- Move in both directions through a result set or answer table
- Order data using an available index
- Work dynamically with the SQL data itself through live access to data sources

See Also

Supported Applications

Supported Databases

Table Requirements

To best take advantage of Borland SQL Link driver capabilities, tables should possess both a unique row identification method and a defined row ordering.

Topics

Unique Row Identification

Defined Row Ordering

Unique Row Identification

Unique row identification is generally recommended for updating data. If the target table contains some non-unique records dBASE cannot absolutely determine which record you want, and as a result the update may fail. Unique row identification improves performance of dBASE DELETE and REPLACE operations, and modifications or deletions made on a form or in BROWSE. (APPEND queries and insertions made using BROWSE or on a form do not require uniquely identifiable rows.)

Borland SQL Links requires some kind of unique row identification to support full BLOB access for SQL servers that do not support BLOB handles for random reads and writes. Most SQL servers limit a single sequential BLOB read to less than the maximum size of a BLOB. In those cases an entire BLOB may not be available. To see if your SQL server tables support BLOB handles, or identify the maximum size of a single BLOB read, see your Connecting to... manual.

You can guarantee unique row identification through either a unique index or an implicit row identification method.

Note: Servers that support implicit row identification do not always support it for all server objects. For example, even if your server supports an implicit unique row identification method for tables, it may not support one for SQL server views. Your Connecting to... manual notes whether your server supports an implicit unique row identification method.

Defined Row Ordering

SQL Links requires a defined row ordering to access a small window of data, centered around the current row location.   The window into the server data moves as the current row location moves, and can be refreshed from the server through the dBASE REFRESH command.

If no defined row ordering is available, when data is inserted, SQL Links cannot tell where the SQL server places the inserted row. Therefore, the row may or may not appear in the set of data as it is read. This behavior can vary from SQL server to SQL server, and even from table to table.

A defined row ordering requires either an index or some other method that identifies individual rows.

SQL Link Locking and Transaction Support

The SQL Link driver enables support for table locking, record locking, and explicit SQL transaction management.   For information on how your SQL server handles locking and transactions, see your SQL server documentation.

Topics

Default Transaction Behavior

SQL operations always take place within the context of a transaction. When no explicit transaction occurs, SQL Links manages the SQL server transactions transparently for the client. Any successful modification of SQL server data is immediately committed to ensure its permanence in the database. For example, a single REPLACE, a single APPEND, and a single form edit operation are each individually committed by default.

Table-locking Behavior

The SQL Link driver provides the same table locking support as the target SQL server. For information on locking support for your SQL server, see your server documentation.

In SQL servers that support table locks, locks can be maintained only within the context of a transaction. A lock is not acquired until after the transaction starts, and can be released only when the transaction ends. When SQL Links acquires a table lock, it automatically starts a transaction if necessary. When SQL Links is ready to release a table lock it first commits the transaction, automatically releasing all other locks at the same time. It then automatically reacquires any remaining locks.

Note: During the period between the time a lock is released and then reacquired, it is possible for another user to change your data. For this reason, it is recommended to either release all table locks together when the last lock is no longer needed, or use explicit SQL transactions instead of locking entire tables.

Record-locking Behavior

SQL servers lock data as required, depending on the type and granularity of lock supported. SQL Links offers an additional locking strategy called optimistic locking to provide a generic way to ensuring data integrity.

Optimistic locking allows a user to modify a local copy of a record, instead of locking a record for the entire time it is being modified. When the modifications are finished, SQL Links checks the current "live" data to make sure no other changes have been made in the interim, then modifies the "live" data based on the changes made to the copy. If the "live" data was changed by someone else, an optimistic lock failure occurs. The user is notified that someone else has changed the data first. They can then inspect the new data and decide whether or not to make changes at that time.

The operation is said to be optimistic because it assumes that no other user will change the record.

Optimistic locking enables users to modify data without the performance and concurrency penalty that comes with locking the data.

Client-controlled Transaction Behavior

SQL Links provides dBASE functions to explicitly begin, commit, and roll back a transaction. This gives the programmer the same protection afforded to pass-through SQL users. When a transaction is explicitly started, a COMMIT operation is never performed automatically, so the client has the full protection of the SQL server transaction mechanism. That is, modifications are performed but not automatically committed, and table lock releases do not cause a commit within an explicit client transaction.

How transactions are started

When starting a transaction, SQL Links uses the same default isolation level in use at the SQL server. For information on the default isolation level of your server, see your server documentation.

An SQL Link transaction is always started and ended through a database alias. Once a transaction is started, all modification and table- and record-locking operations executed in that database are performed in the context of that transaction.

How SQL Link operations change under an explicit transaction

When operating within an explicit client transaction, some SQL Link operations cannot be made atomic. Both table append operations and locally-executed queries execute in their own transaction by default. When executed within an explicit client transaction, they execute within the client transaction context. If such an operation fails during processing, the operation may be only partially complete.

Although transactions can be started in multiple database aliases simultaneously, SQL Links manages the transactions independently, so the user must be careful to minimize the window of vulnerability after the first commit of a multiple database transaction.

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

SQL-enabled dBASE

See Also

dBASE provides new commands that support SQL and transaction processing. If you are familiar with SQL, you can also include SQL statements in dBASE code using SQLEXEC().

This section provides information on dBASE commands compatible for use in an SQL environment. For detailed information on dBASE commands, see the dBASE for Windows Programmer's Guide or dBASE for Windows online Help.

Topics

See Also

Working With SQL Link Drivers

Creating dBASE Applications for SQL Data

dBASE Language Extensions for SQL

SeeAlso

This section briefly summarizes dBASE commands that support SQL and transaction processing. For detailed information on dBASE commands (including complete syntax and further examples), see the dBASE for Windows Programmer's Guide or dBASE for Windows online Help.

Topics

BEGINTRANS()

COMMIT()

OPEN DATABASE

ROLLBACK()

SQLEXEC()

See Also

Using Transactions in dBASE Applications

BEGINTRANS()

Example

Starts a transaction.   Returns .T. if the transaction started successfully.

Use BEGINTRANS() to initiate a transaction during which the user might make changes to an SQL database that supports transaction processing.

For more detailed information, see the dBASE for Windows Programmer's Guide or online Help.

Syntax

BEGINTRANS ([<database name expC>])

<database name expC>
The name of the SQL database in which to begin the transaction.   If <database name expC> is omitted but a SET DATABASE statement has been issued, BeginTrans() refers to the database in the SET DATABASE statement.   If <database name expC> is omitted and no SET DATABASE statement has been issued, BeginTrans() refers to the database opened after issuing BeginTrans().

Limitations

Nested transactions are not permitted.

Transaction Example

The following example shows how you might define a transaction for data stored on an SQL server:

```
ON ERROR DO errorProc                        && Specify error processing routine
OPEN DATABASE AcctgSybase                           && Open a database
USE :AcctgSybase:mytable                            && Open a table in the database
IF BEGINTRANS()                                     && Start a server transaction
  REPLACE ALL Salary WITH Salary * Inflation        && Try to update all records
  IF .NOT. COMMIT()                                 && Commit changes if no error occurred
    *Commit failed
    :                                               && Provide user options in case the
    .                                               && transaction fails
    ENDIF
ENDIF
  :
  .
PROCEDURE errorProc                                 && Define procedure for handling errors
  IF .NOT. ROLLBACK()                               && Try to roll back if error occurred
    * Rollback failed
    IF DBERROR() <>0                                && Check for server error
      * Process IDAPI Error
      :
      .
      IF SQLERROR() <>0                             && Check for SQL server error
        * Process SQL Error
      ENDIF
    ENDIF
  ENDIF
RETURN
```

COMMIT()

Example

Ends a transaction initiated by BEGINTRANS() and writes to the open files any changes made during the transaction. Returns .T. if the data was committed successfully.

To end an open transaction without writing changes to the file, use ROLLBACK().

For more detailed information, see the dBASE for Windows Programmer's Guide or online Help.

Syntax

COMMIT ([<database name expC>])

<database name expC>

The name of the SQL database in which to complete the transaction. If you began the transaction with BEGINTRANS <database name expC>, you must specify <database name expC> when you COMMIT. If you began the transaction with BEGINTRANS(), you can omit <database name expC>.

OPEN DATABASE

Example

Opens a STANDARD or SQL server alias defined through the IDAPI Configuration Utility. Required to establish a connection with an SQL database before you can access its tables. If you eliminate the LOGIN clause, dBASE displays a login dialog.

Note: You can also use a database qualifier to open or create tables in databases, overriding the default SET DATABASE TO and SET DBTYPE TO settings. For example,
USE :SYBASE1:mayors, where SYBASE1 is the database name and mayors is the table name.

Syntax

```
OPEN DATABASE <database name>
   [LOGIN <username>/<password>]
   [WITH <option expC>]
```

<database name>

The name of the database you want to open. Database aliases are created using the IDAPI Configuration Utility (see Getting Started for more information).

<user name>/<password>

Character string specifying the user name and password combination required to access the database.

WITH <option string>

Character string specifying server-specific information required to establish a connection with an SQL server. For information about establishing SQL server connections, see your Borland SQL Link Getting Started manual.

OPEN DATABASE Example

The following example uses OPEN DATABASE to establish a connection with an SQL server, opens a database previously created using the IDAPI Configuration Utility with SET DATABASE TO, and opens a server dBASE table and appends it to the client/server database:

```
OPEN DATABASE CAClients                    && Establish connection with database server
USE :CAClients:Company                     && Opens server table named Company
APPEND FROM CLIENTS.DBF                    && Appends data from local dBASE table
CLOSE DATABASE CAClients                   && Closes Company and disconnects
                                           && from database server
```

ROLLBACK()

Example

Ends a transaction initiated by BEGINTRANS() without saving any changes to the open files.   Returns .T. if the data was committed successfully.

For more detailed information, see the dBASE for Windows Programmer's Guide or online Help.

Syntax

ROLLBACK ([<database name expC>])

<database name expC>

The name of the SQL database in which to cancel the transaction.   If you began the transaction with BEGINTRANS <database name expC>, you must specify <database name expC> when you roll back the transaction.   If you began the transaction with BEGINTRANS(), you can omit <database name expC>.

SQLEXEC()

Examples

Allows SQL language to be used on local and server tables. This statement returns an error code referencing the same error numbers that the dBASE for Windows ERROR() and MESSAGE() error codes use. (0 means no error occurred.)

For more detailed information, see the dBASE for Windows Programmer's Guide or online Help.

Syntax

SQLEXEC(<SQL statement expC1> [, <Answer table expC2>])

<SQL statement expC1>

A character string that contains an SQL statement. The statement must follow server-specific dialect rules and must be enclosed in quotes. If the SQL statement includes character strings or SQL or IDAPI reserved words, these must also be enclosed in either single or double quotation marks.

The database against which the SQL statement is executed is based on the setting of SET DATABASE TO. If SET DATABASE is not set, then the SQL statement is operating on a local table (dBASE or Paradox only).

<Answer table expC2>

An optional dBASE or Paradox answer table that stores the data returned by an SQL SELECT statement. The type of table returned is based on the setting of SET DBTYPE TO. Display of SQL server answer tables is not supported. If you do not specify a table name, dBASE creates a table named Answer with the appropriate extension.

SQLEXEC() Examples

The following example shows table creation with SQLEXEC() and native SYBASE SQL data types.
:SYBASE: is a Sybase alias in the IDAPI Configuration Utility.:

```
OPEN DATABASE SYBASE                        && login to Sybase
SET DATABASE TO SYBASE                      && focus on Sybase alias

** Create CUSTOMER table on Sybase

SQLString = "create table CUSTOMER (Client_ID int not null, ";
  + "AcctNum char(8) not null,Client_Name varchar(20),Balance money)"

SQLIndex = "create unique clustered index;
    CLIENT_IDX_ID_NAME_U on ";
    + " CUSTOMER(CLIENT_ID,CLIENT_NAME)"

Ret = SQLEXEC(SQLString)                    && create the table from SQL statement
                                            && Sybase will automatically commit the DDL
IF Ret = 0
  Ret = SQLEXEC(SQLIndex)                   && create unique clustered Sybase index
ENDIF

** Create ORDERS table on Sybase

SQLString = "create table ORDERS(Order_ID int not null, ";
    + "Order_Date datetime not null,Part_ID char(10),
    Order_Status char(1))"

Ret = SQLEXEC(SQLString)                    && create the table from SQL statement
                                            && Sybase will automatically commit the DDL

SQLIndex1 = "create unique clustered index;
    ORDER_IDX_ID_DATE_U on ";
    + " ORDERS(Order_ID,Order_Date)"

IF Ret = 0
  Ret = SQLEXEC(SQLIndex1)                  && create unique clustered Sybase index
ENDIF

SQLIndex2 = "create index ORDER_IDX_STATUS;
  on ORDERS(Order_Status)

IF Ret = 0
  Ret = SQLEXEC(SQLIndex2)                  && create Sybase index
ENDIF


 SET DATABASE TO                            && remove focus
 CLOSE DATABASE SYBASE                      && close Sybase alias
```

The following example shows table creation with SQLEXEC() and native ORACLE SQL data types.
:ORACLE: is an Oracle alias in the IDAPI Configuration Utility.

```
OPEN DATABASE ORACLE                        && login to ORACLE
SET DATABASE TO ORACLE                      && focus on ORACLE alias

** Create CUSTOMER table on ORACLE

SQLString = "create table CUSTOMER (Client_ID NUMBER(15) not null, ";
    + "AcctNum VARCHAR2(8) not null,Client_Name VARCHAR2(20),Balance NUMBER(16,2)"

SQLIndex = "create unique index CLIENT_IDX_ID_NAME_U on ";
    + " CUSTOMER(CLIENT_ID,CLIENT_NAME)"

Ret = SQLEXEC(SQLString)                    && create the table from SQL statement
                                            && ORACLE will automatically commit the DDL
IF Ret = 0
  Ret = SQLEXEC(SQLIndex)                   && create unique clustered ORACLE index
ENDIF

** Create ORDERS table on ORACLE

SQLString = "create table ORDERS(Order_ID NUMBER(15) not null, ";
    + "Order_Date DATE not null,Part_ID VARCHAR2(10),Order_Status VARCHAR2(1))"

Ret = SQLEXEC(SQLString)                    && create the table from SQL statement
                                            && ORACLE will automatically commit the DDL

SQLIndex1 = "create unique index ORDER_IDX_ID_DATE_U on ";
    + " ORDERS(Order_ID,Order_Date)"

IF Ret = 0
  Ret = SQLEXEC(SQLIndex1)                  && create unique ORACLE index
ENDIF

SQLIndex2 = "create index ORDER_IDX_STATUS on ORDERS(Order_Status)

IF Ret = 0
  Ret = SQLEXEC(SQLIndex2                   && create ORACLE index
ENDIF


SET DATABASE TO                             && remove focus
CLOSE DATABASE ORACLE               && close ORACLE alias
```

SQL Error-handling in dBASE

Example

dBASE for Windows dedicates two error codes to retrieving the number and text of SQL server messages.

For a complete list of all dBASE for Windows error codes, see the dBASE for Windows Programmer's Guide or online Help.

SQLERROR()
Returns the number of the last server error.

Syntax
SQLERROR()

SQLMESSAGE()
Returns the text of the last server error.

Syntax
SQLMESSAGE()

Other error codes that are especially useful for detecting and handling broken record and file locks are

DBERROR() - returns IDAPI error number
DBMESSAGE() - returns IDAPI error message
ERROR() - returns dBASE error number
MESSAGE() - returns dBASE error message

SQL Error-handling Example

The following example uses SQLERROR( ) and SQLMESSAGE( ) to return an SQL error number and SQL error message to an ON ERROR routine that displays a MDI form with an error report:

```
ON ERROR DO ErrHndlr WITH ERROR(), MESSAGE(), ;
  SQLERROR(), SQLMESSAGE(), PROGRAM(), LINENO()
SET DBTYPE TO DBASE
OPEN DATABASE CAClients
errorCode = SQLEXEC("SELECT Company, City FROM ;
  Company WHERE State_Prov='CA'", "StateCA.DBF")
IF errorCode = 0
  SET DATABASE TO
  USE StateCa
  LIST
ENDIF
RETURN

PROCEDURE ErrHndlr
PARAMETERS nErrorNo, cErrMess, nSQLErrorNo, ;
  cSQLErrMess, cProgram, nLineNo
DEFINE FORM HeadsUp FROM 10,20 TO 20,55;
  PROPERTY Text "Heads Up"
DEFINE TEXT Line1 OF HeadsUp AT 2,10 ;
  PROPERTY Text "An Error has occurred",;
  Width 24, ColorNormal "R+/W"
DEFINE TEXT Line2 OF HeadsUp AT 4,2;
  PROPERTY Text ;
  IIF(ERROR()=240,cSqlErrMess,cErrMess),;
  Width 33
DEFINE TEXT Line3 OF HeadsUp AT 5,2;
  PROPERTY Text "Number: " + ;
  IIF(ERROR()=240,STR(nSQLErrorNo),STR(nErrorno)),;
  Width 24
DEFINE TEXT Line4 OF HeadsUp AT 6,2;
  PROPERTY Text "Program: "+ cProgram,;
  Width 22
DEFINE TEXT Line5 OF HeadsUp AT 7,2;
  PROPERTY Text "Line #: " + STR(nLineno),;
  Width 22
OPEN FORM HeadsUp
```

Composite Keys

This release of dBASE for Windows only allows expressions in index keys. Only single or composite keys are allowed for other drivers, including the SQL drivers. The commands that support composite keys are:

```
INDEX ON
SEEK
SET KEY TO
SET RELATION TO
```

Other dBASE Language Elements That Support SQL

All dBASE for Windows data access comands and functions support SQL data.   Some of the most useful are:

APPEND	DELETE TAG
APPEND FROM	DELETE TABLE
BOOKMARK()	GO TO <bookmark>
BROWSE	INDEX
CLOSE DATABASE	REPLACE
COPY TABLE	SEEK
COPY TO	SET DATABASE TO
CREATE	SET RELATION
DELETE	

For further information on these commands, see the dBASE for Windows Programmer's Guide or online Help.

Creating dBASE Applications for SQL Data

Collapse List
See Also

Topics

Creating dBASE Applications for SQL Data

Expand list
See Also

Topics

See Also

Working with SQL Link drivers

SQL-enabled dBASE

Moving a Local Application to the SQL Environment

Topics

Tips For Converting dBASE Applications

- Use COPY TO to export existing tables.
- Add OPEN DATABASE and SET DATABASE TO to the application program.
- Add error handling for REPLACE errors (broken locks) with ON ERROR and/or DBERROR().
- If desired, add BEGINTRANS(), COMMIT() and ROLLBACK() to each transaction sequence.
- Replace any use of RecNo() with Bookmark().
- Use SQLEXEC() to create any new tables with native SQL data types.

Note: New and existing dBASE for Windows applications that use the RLOCK function will need additional error-checking when used with SQL databases.

Things to Know About the Target SQL Server

See Also

Item	InterBase	Oracle	Sybase
SQL Link driver Dynamic Link Library (DLL) name	SQLD_IB.DLL	SQLD_ORA.DLL	SQLD_SS.DL L
Case-sensitive for data?	Yes (including pattern matching)	Yes	As installed
Case-sensitive for objects such as tables, columns, indexes?	No	No	As installed
Does the server require that you explicitly start a transaction for multistatement transaction processing?	Yes	Yes	Yes
Does the server require that you explicitly start a transaction for multistatement transaction processing in pass-through SQL?	No	No	Yes
Implicit row IDs?	No	Yes	No
BLOB handles?	InterBase BLOBs have handles. However, InterBase CHAR and VARCHAR columns that are more than 255 characters long are treated as non-handle BLOBs.	No	No
Maximum size of single BLOBs read (if BLOB handles are not supported)	32K	64K	32K

See Also

InterBase Data Type Translations

Oracle Data Type Translations

Sybase Data Type translations

InterBase Data Type Translations

FROM: InterBase	TO: Paradox	dBASE	Oracle	Sybase
Short	Short	Number{6.0}	Number	SmallInt
Long	Number	Number{11.0}	Number	Int
Float	Number	Float{20.4}	Number	Float
Double	Number	Float{20.4}	Number	Float
Char	Alpha	Character	Character	VarChar
Varying	Alpha	Character	Character	VarChar
Date	DateTime	Date	Date	DateTime
Blob	Binary	Memo	LongRaw	Image
Blob/1	Memo	Memo	Long	Text

Oracle Data Type Translations

FROM: Oracle	TO: Paradox	dBASE	Sybase	InterBase
Character	Alpha	Character	VarChar	Varying
Raw	Binary	Memo	VarBinary	Varying
Date	DateTime	Date	DateTime	Date
Number	Number	Float{20.4}	Float	Double
Long	Memo	Memo	Text	Blob/1
LongRaw	Binary	Memo	Image	Blob

Sybase Data Type Translations

FROM: Sybase	TO: Paradox	dBASE	Oracle	InterBase
Character	Alpha	Character	Character	Varying
VarCharacter	Alpha	Character	Character	Varying
Int	Number	Number{11.0}	Number	Long
SmallInt	Short	Number{6.0}	Number	Short
TinyInt	Short	Number{6.0}	Number	Short
Float	Number	Float{20.4}	Number	Double
Money	Money	Float{20.4}	Number	Double
Text	Memo	Memo	Long	Blob/1
Binary	Binary	Memo	Raw	Varying
VarBinary	Binary	Memo	Raw	Varying
Image	Binary	Memo	LongRaw	Blob
Bit	Alpha	Bool	Character	Varying
DateTime	DateTime	Date	Date	Date
TimeStamp	Binary	Memo	Raw	Varying
Float4	Number	Number	Number	Double
Money4	Money	Float{20.4}	Number	Double

DateTime4	DateTime	Date	Date	Date

Multiuser Considerations

dBASE uses record locking (RLOCK) to help ensure data integrity in a multiuser environment; Paradox uses both record and file locking (FLOCK). As noted in SQL Link Locking and Transaction Support, however, SQL does not implement locking. This means that every record update must be checked with ON ERROR or DBERROR() to determine if a lock was broken prior to completion of the transaction. If the user attempts an APPEND, BROWSE, or EDIT and the record lock breaks, dBASE for Windows displays a dialog box warning the user of the problem. The user can then retry the update.

Blank and Duplicate Record Handling

In general, SQL databases that use a primary or uniquely-keyed index do not allow blank and duplicate records. If the user attempts to create such a record with APPEND, BROWSE or EDIT, dBASE for Windows displays a dialog box warning the user of the problem. The user can then retry the update. If the user attempts to create a blank or duplicate record outside APPEND, BROWSE or EDIT, dBASE for Windows issues an error.

Using Transactions

See Also

dBASE for Windows' file-based method of database management differs significantly from SQLs transaction-oriented method.

In dBASE changes, additions, and deletions of records are made to the actual tables in which the data are stored. Record and table (file) locks are applied to insure data integrity by keeping more than one user from modifying the same record at the same time.

In SQL, the user requests a record or set of records, which are then transparently copied and made available to the user. Changes, additions, and deletions are made to the copy of the data, and only made permanent (committed) when the transaction is complete. The transaction model includes the ability to apply record and table locks, depending on what the SQL server will support; however, it is the isolation of the transaction itself that ensures data integrity.

dBASE for Windows implements the dBASE model of direct data manipulation for local data, but also provides functions that can work in the transaction-oriented format required by SQL when a supported Borland SQL Link driver is installed. dBASE for Windows supports transactions against both local (dBASE, Paradox) and SQL server data.   It processes local transaction operations itself and passes server transactions to the SQL server to be processed there.

dBASE for Windows replaces the traditional dBASE transaction model with new event-oriented transaction functions.   The new transaction model supports SQL transactions by:

- Adding new language elements and transaction functions such as BEGINTRANS(), COMMIT(), and ROLLBACK()
- Using a ROLLBACK() that does not change program flow
- Not supporting the USE NOSAVE option
- Modifying CANCEL so that it does not rollback transactions
- Not tracking APPEND FROM

Note: You cannot mix local transactions and SQL server transactions. Once you start a local transaction with BEGINTRANS() you cannot start an SQL server transaction until you either COMMIT() or ROLLBACK().   Any such attempt will display an error message.

To create forms that use transactions

- Include BEGIN TRANS() in the startup code for a form or as pushbutton for "Starting Changes".
- Include COMMIT() in the OnClick event handler for "Finished Changes".
- Attach ROLLBACK() to a pushbutton for "Cancel Changes".
- Use COMMIT() or ROLLBACK() in the ON CLOSE() event handler to clean up when the form is closed.   If a transaction is not ended properly (by either a COMMIT() or a ROLLBACK()), dBASE displays the Transaction Active dialog

Note: The dBASE IV BEGIN TRANSACTION, END TRANSACTION, and ROLLBACK commands are non-operational and return a warning.

See Also

Table-locking

Record-locking

Client-controlled transaction behavior

Pass-through SQL

The pass-through SQL feature enables users to "pass" SQL statements directly to the SQL server using the dBASE SQLEXEC() function. Unlike queries made through your dBASE for Windows desktop, pass-through SQL queries are not filtered through the IDAPI database engine and translated into the appropriate SQL dialect. Pass-through statements must use the same semantics as statements entered directly as the SQL server.

Pass-through SQL is considered to be a separate kind of connection from the conventional desktop application connection. However, your alias SQLPASSTHRU MODE can be set to allow desktop commands and pass-through SQL statements in the same alias connection. For further information, see the IDAPI Configuration Utility online Help or your SQL Links Getting Started manual.

Transaction

A group of related operations that must all be performed successfully before the RDBMS will finalize any changes to the database.

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

Atomicity

Atomicity is the "all-or-nothing" characteristic of transactions.

All operations contained in a transaction are said to occur within the context of that transaction. The transaction succeeds only if every operation, no matter how insignificant it may seem, succeeds. If any operation within the context of a transaction fails, the entire transaction fails.

Supported Borland applications

Borland SQL Links works with any application that supports IDAPI, the Borland Integrated Database Application Programming Interface).   This includes Paradox for Windows, dBASE for Windows, and custom IDAPI applications.

Supported databases

Drivers in the SQL Links product package support InterBase, Informix, ORACLE, and SYBASE and Microsoft SQL Server databases. dBASE for Windows works with the SQL Link InterBase, ORACLE, and SYBASE drivers.

Heterogeneous queries

Queries that involve more than one table type; for example, a Paradox table, a dBASE table, and an InterBase table.