Introduction to Upsizing Your *Visual* dBASE Application

Migrating a desktop application to a client/server environment is called upsizing. This Help system is designed to familiarize the *Visual* dBASE user with some of the more important upsizing aspects:

Understanding Connectivity Issues

Upsizing the Database

Understanding Connectivity Issues

Borland SQL Links for Windows is a collection of drivers that enable you to connect through the Borland Database Engine (BDE) to remote database servers: Oracle, Sybase, Informix, Microsoft SQL Server, and InterBase.

The SQL Link driver provides the connection to the SQL server, translates queries into the appropriate SQL dialect, and passes them to the SQL database server. When processing is complete, the SQL database returns the answer to the client in a format that the desktop application can display.

Note: Borland database applications, through the BDE, also support the use of SQL statements against local (Paradox or dBASE) data. For information on how to use Local SQL with *Visual* dBASE, see the *Visual* dBASE online Help and print documentation.

Through the BDE Configuration Utility, you can set up an alias for each data source to which your application connects. *Visual* dBASE client applications can use any network protocol (such as TCP/IP, Novell SPX/IPX, or NetBEUI) supported by the server, as long as both the server and the client machines have the proper communication software installed. You must configure the SQL Link driver for the desired protocol.

For more information about connectivity, see the *SQL Links for Windows User's Guide* and the help provided with SQL Links and the BDE Configuration Utility.

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

Unique Row Identification

Unique row identification is generally recommended on SQL database servers that dBASE accesses. If the target table contains non-unique records, dBASE cannot absolutely determine which record to update and the update may fail. Unique row identification can improve performance of dBASE DELETE and REPLACE operations, in addition to modifications or deletions made on a form or using BROWSE. (APPEND queries and insertions made using BROWSE or on a form do not require uniquely identifiable rows.)

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.

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:

- If your target SQL server supports PRIMARY KEYs, you can create the key using *Visual* dBASE's Define Primary Key Dialog Box in the Table Designer or the SQL command ALTER TABLE.
- Unique indexes are created using the Manage Indexes Dialog Box, the dBASE INDEX command, or the SQL command CREATE INDEX.

Note: Since Oracle databases support a unique row ID, a unique index is not required.

Defined Row Ordering

SQL Links requires 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.

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

Upsizing the Database

Upsizing requires a shift in perspective from the desktop world to the client/server world. To fully understand the subject requires not only an understanding of the differences between the database features, but how and where processing takes place.

For example, dBASE databases are conceptually record-oriented, while SQL database servers are conceptually set-oriented. dBASE databases typically store each table in a separate file, while servers store all the tables in a database together. On SQL database servers, all processing takes place on the server in a multi-server environment.

Client/server applications must also address some database issues in an entirely new manner, the most complex of which are connectivity, network usage, and transaction handling. For more information, see your SQL database documentation and any of the texts written on these subjects.

Upsizing a database includes the following:

- Analyzing the differences between dBASE data and server data
- Migrating the data from the desktop to the server
- Using the Local InterBase Server (LIBS)

Analyzing the Differences Between Databases

As part of the process of upsizing data from dBASE to an SQL database server, the following issues must be understood:

Data Type Translations

Referential Integrity

Transaction Control

Data Access Rights

Indexing

Stored Procedures and Triggers

Data Security

Target Server Information

Data Type Translations

Physical data type translations from dBASE tables to other server types:

From dBASE	To Paradox	To Oracle	To Sybase	To InterBase	To Informix
Character	Alpha	Character	VarChar	Varying	Character
Number	Short	Number	SmallInt	Short	SmallInt
others	Number	Number	Float	Double	Float
Float	Number	Number	Float	Double	Float
Date	Date	Date	DateTime	Date	Date
Memo	Memo	Long	Text	Blob/1	Text
Bool	Bool	Character{1}	Bit	Character{1}	Character
dbaselock	Alpha{24}	Character{24}	Character{24}	Character{24}	Character
OLE	OLE	LongRaw	Image	Blob	Byte
Binary	Binary	LongRaw	Image	Blob	Byte
Bytes	Bytes	LongRaw	Image	Blob	Byte (temp tables)

Referential Integrity

In SQL databases, integrity constraints are rules that govern columns-to-table and table-to-table relationships and validate data entries. They span all transactions that access the database and are automatically maintained by the system. Integrity constraints can be applied to an entire table or to an individual column. A PRIMARY KEY or Unique constraint guarantees that no two values in a column or set of columns will be the same.

For information about SQL-database server integrity constraints, refer to your server documentation or the online help provided with the Local InterBase Server.

dBASE lets you establish integrity constraints for any file type that supports it using the Database Administration dialog box and choosing Referential Integrity.

Column constraints can be added through the Table Designer.

For more information on integrity constraints in *Visual* dBASE, refer to the *User's Guide*.

Transaction Control

SQL database servers handle requests in logical units of work called transactions. A transaction is a group of SQL statements that must all be performed successfully before the server will finalize (or commit) changes to the database. Either all the statements will succeed, or all will fail.

Transaction processing ensures database consistency even if there are hardware failures and maintains the integrity of data while allowing concurrent multiuser access. For example, an application might update the ORDERS table to indicate that an order for a certain item was taken and then update the INVENTORY table to reflect the reduction in inventory available. If there were a hardware failure after the first update but before the second, the database would be in an inconsistent state, since the inventory would not reflect the order entered. Under transaction control, both statements would be committed at the same time. Transaction control becomes even more important in a multiuser application.

In SQL, transactions are explicitly ended with a command to either accept or discard the actions performed. The COMMIT statement permanently commits the transaction, making changes visible to all users. The ROLLBACK statement undoes all changes made to the database in the transaction. Different database servers implement transaction processing differently. For the specifics of how your server handles transaction processing, refer to your server documentation.

Important areas for understanding transactions:

- dBASE transaction management
- Transaction isolation levels
- Transaction control through the BDE

dBASE Transaction Management

Visual dBASE's 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.

SQL operations always take place within the context of a transaction. 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 locks and table locks, depending on what the SQL server will support. However, it is the isolation of the transaction itself that ensures data integrity.

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.

Visual dBASE implements the dBASE model of direct data manipulation for local data. It also provides functions that can work in the transaction-oriented format required by SQL, when a supported Borland SQL Link driver is installed. *Visual* dBASE 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.

Visual dBASE 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
- 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 BEGINTRANS( ) 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 box.

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 and automatically releases all other locks at the same time. It then automatically re-acquires any remaining locks.

Note: During the period between the time a lock is released and then re-acquired, 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.

Transaction Isolation Levels

A transaction's isolation level determines how it interacts with other simultaneous transactions accessing the same tables. In particular, the isolation level affects what a transaction reads from the tables being accessed by other transactions.

Some servers enable you to set the transaction isolation level explicitly in passthrough SQL. If not specified, passthrough SQL operations will use a server's default isolation level. For more information, see your server documentation.

dBASE allows you to define the server-level transaction isolation scheme within BEGINTRANS( ), as follows:

Option 0 Use the server-level default isolation level

Option 1 DirtyRead: The transaction can read uncommitted changes to the database by other transactions. This is the lowest isolation level.

Option 2 ReadCommitted: The transaction can read only committed changes to the database by other transactions. This is the default isolation level.

Option 3 RepeatableRead: The transaction cannot read other transactions' changes to previously read data. This guarantees that once a transaction reads a record, it will not change if it reads it again. This the highest isolation level.

Database servers may support these isolation levels differently or not at all. If the requested isolation level is not supported by the server, the next highest isolation level is used. The actual isolation level used by each type of server is shown below:

Isolation setting	Oracle	Sybase and Microsoft SQL servers	Informix	InterBase
Dirty read	Read committed	Read committed	Dirty read	Read committed
Read committed (Default)	Read committed	Read committed	Read committed	Read committed
Repeatable read	Repeatable read (READ ONLY)	Read committed	Repeatable read	Repeatable read

Note: If an application is using ODBC to interface with a server, the ODBC driver must also support the isolation level. For more information, see your ODBC driver documentation.

Transaction Control Through the BDE

SQLPASSTHRUMODE in the BDE Configuration Utility determines if passthrough SQL and standard BDE calls share the same database connection. For transactions, this translates to whether passthrough transactions and other transactions "know" about each other. Only applications that use passthrough SQL need to be concerned with SQLPASSTHRUMODE.

SQLPASSTHRUMODE can have the following settings:

- SHARED AUTOCOMMIT Each operation on a single row is committed. This mode most closely approximates desktop database behavior, but is inefficient on SQL servers because it starts and commits a new transaction for each row, resulting a heavy load of network traffic.
- SHARED NOAUTOCOMMIT The application must explicitly start and commit transactions. This setting can result in conflicts in busy, multiuser environments, where many users are updating the same rows.
- NOT SHARED Passthrough SQL and dBASE use separate database connections.

Note: To control transactions with passthrough SQL, you must set SQLPASSTHRUMODE to NOT SHARED. Otherwise, passthrough SQL and dBASE may conflict.

Data Access Rights

On SQL database servers, a user name and password are usually required to log on to the server. To access data created on an SQL server from dBASE you must:

1. Define an alias for the database using the BDE Configuration Utility.
2. Use the alias to access the database. If you use OPEN DATABASE to access the database, *Visual* dBASE displays a dialog box in which you enter your user name and password. User authentication is usually created at the server level. For information about user authentication, refer to your server documentation.

Indexing

On SQL database servers, an index is a mechanism that is used to speed the retrieval of records in response to certain search conditions and to enforce uniqueness constraints on columns. It serves as a logical pointer to the physical location (address) of a row in a table. An index stores each value of the indexed column or columns along with pointers to all of the disk blocks that contain rows with that column value. On SQL database servers, unique indexes are defined for a table as PRIMARY KEY and FOREIGN KEY constraints.

Note: Upsizing your *Visual* dBASE application requires converting indexes dealing with unsupported index keys such as expression index keys, filters, and UDF's in keys.

For information about how indexes are used in SQL databases, refer to your server documentation. The *Visual* dBASE *Programmer's Guide* discusses working with indexes on non-dBASE tables.

For information about using indexes on tables created in *Visual* dBASE, refer to the *User's Guide* and online Help.

Stored Procedures and Triggers

On SQL databases, a stored procedure is a self-contained server-based program that can take input parameters and return output parameters to an application. Stored procedures are associated with a database and are actually part of metadata. For information about how stored procedures are called from *Visual* dBASE applications, refer to the *Programmer's Guide* or the online Help.

On SQL databases, a trigger is a self-contained routine associated with a table or view that automatically performs an action when a row in a table or view is inserted, updated, or deleted. A trigger is never called directly. When an application or user attempts to INSERT, UPDATE, or DELETE a row in a table, any trigger associated with that table and operation is automatically executed or fired.

For information about triggers, refer to your server documentation.

Data Security

In SQL databases, SQL security is controlled at the table level with access privileges, a list of operations that a user is allowed to perform on a given table or view. The GRANT statement assigns access privileges for a table or view to specified users or procedures. The REVOKE statement removes previously granted privileges. *Visual* dBASE allows you to use the GRANT and REVOKE statements within the dBASE language.

For information about granting and revoking table access on SQL servers, refer to your server documentation.

Target SQL Server Information

Item	InterBase	Oracle	Sybase
SQL Link driver Dynamic Link Library (DLL) name	SQLD_IB.DLL	SQLD_ORA.DLL	SQLD_SS.DLL
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 multi-statement transaction processing?	Yes	Yes	Yes
Does the server require that you explicitly start a transaction for multi-statement transaction processing in pass-through SQL?	No	No	Yes
Implicit row IDs?	No	Yes	No
BLOB handles?	Yes	No	No
Maximum size of single BLOBs read (if BLOB handles are not supported)	32K	64K	32K

Note: InterBase BLOBs have handles. However, InterBase CHAR and VARCHAR columns that are more than 255 characters long are treated as non-handle BLOBs.

Migrating the Data

The following topics provide tips for dBASE developers who want to migrate to an SQL environment:

Moving the Data

Recreating Your Data Models

Creating Tables on the Server Using *Visual* dBASE

Establishing a Database Connection in Your Application

Establishing Explicit Transaction Control

Visual dBASE Language Extensions

Using SQL Syntax Within *Visual* dBASE

Visual dBASE Language Elements that Support SQL Data

SQL Error-handling in dBASE

Multiuser Considerations

Blank and Duplicate Record Handling

Other Considerations

Moving the Data

To copy single dBASE tables to an SQL database server, you can use COPY. The COPY command not only creates table structures on the database server, but copies records from the source table to the destination database server.

Batch moves can be performed using APPEND FROM. After server data is established, you can perform batch uploads to SQL data from dBASE files.

The fastest and most efficient tool for uploading data is the Borland Data Pump. For information about using the Data Pump, refer to the Data Pump online Help.

Recreating Your Data Models

You can use the *Visual* dBASE Query Designer to recreate your data models after you have migrated your data. This allows Forms and Reports that are based on query files to run unmodified.

Creating Tables on the Server Using *Visual* dBASE

The Table Designer provides a method of creating single tables on SQL database server. The Table Designer provides access to creating all field types supported on the server.

Establishing a Database Connection in Your Application

To establish a database connection in your application you must first create an alias for the database using the BDE Configuration Utility and then add OPEN DATABASE and SET DATABASE to the application program.

Establishing Explicit Transaction Control

To establish explicit transaction control, add BEGINTRANS( ), COMMIT( ), and ROLLBACK( ) to each transaction sequence.

Visual dBASE Language Extensions for SQL

The following *Visual* dBASE commands support SQL and transaction processing.

Topics

BEGINTRANS( )

COMMIT( )

EXTERN SQL

OPEN DATABASE

ROLLBACK( )

SQLERROR( )

SQLEXEC( )

Using SQL Syntax Within *Visual* dBASE

The following list of SQL statements are supported within *Visual* dBASE:

- ALTER TABLE
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW*
- DELETE
- DROP INDEX
- DROP TABLE
- DROP VIEW*
- GRANT*
- INSERT
- REVOKE*
- SELECT
- SET TRANSACTION*
- UPDATE

*Not supported for local SQL

Visual dBASE Language Elements That Support SQL

All *Visual* dBASE data access commands and functions support SQL data. Some of the most useful are:

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

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

SQL Error-handling in *Visual* dBASE

Example

Visual dBASE dedicates two error codes to retrieve the number and text of SQL server messages.

For a complete list of all *Visual* dBASE error codes, see the *Visual* dBASE *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 BDE error number

DBMESSAGE() - returns BDE 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 an 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
```

Multiuser Considerations

See Also

dBASE can use record locking (RLOCK) to help ensure data integrity in a multiuser environment. 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, *Visual* dBASE displays a dialog box warning the user of the problem. The user can then retry the update.

See Also

Table-locking Behavior

Record-locking Behavior

Blank and Duplicate Record Handling

In general, SQL database servers that use a primary or uniquely-keyed index do not allow blank or duplicate records. Primary keys do not allow NULL records (not blank), but unique keys allow unlimited NULL records. Since BLANK and APPEND for servers default to NULL data, no errors should be encountered on a unique key; however, they would occur on a primary key. If the user attempts to create such a record with APPEND, BROWSE or EDIT, *Visual* dBASE 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, *Visual* dBASE issues an error.

Other Considerations

Add error handling for REPLACE errors (broken locks) with ON ERROR and/or DBERROR( ).

Replace any use of RECNO( ) with BOOKMARK( ).

Determine what dBASE file structures are unsupported on the target server; for example, multiple memos are unsupported on an Oracle table.

New and existing *Visual* dBASE applications that use the RLOCK( ) function will need additional error checking when used with SQL databases.

Using the Local InterBase Server

The Local InterBase Server (LIBS) is a version of the Borland InterBase Workgroup server that runs on Windows 3.1. It provides all of the features of a SQL server for local, single-user operation. For more information on the Local InterBase Server, see the *Local InterBase Server User's Guide*.

The Local InterBase Server can be for client/server development as:

A local environment for building an application to access any server.

An intermediate step for building an application to access the InterBase Workgroup server.

Building an Application to Access Any Server

If you are developing a client/server application, you can use the Local InterBase Server (LIBS) for local development and prototyping, even if the production database will run on another server.

If the production database already exists on a database server, you can use the Local InterBase Server as follows:

1. Use your server's tools to create an SQL data definition script for the database. Remove any non-standard SQL syntax that will not work with InterBase. Generally, this means removing stored procedure definitions and other advanced features and mapping data types to InterBase data types.
2. Use the LIBS tool, Windows ISQL, to create a local InterBase database then execute the SQL script to define the database. For information on using Windows ISQL, see the *Local InterBase Server User's Guide* or the online help provided with Windows ISQL.
3. Populate the database with representative data using dBASE methods or the Borland Data Pump.
4. Create a dBASE database application that uses the Local InterBase Server database as a data source.

If the production database does not already exist, you can first define a prototype database on LIBS and develop the application locally. Simultaneously, you can define a congruent database on the target server. Finally, you can redirect the application to access the database on the target server.

Building an Application to Access InterBase

To build an application that accesses a production database that already exists on an InterBase Workgroup server:

1. Use the InterBase database backup utility to create a backup of the database, using the "Backup Metadata Only" option. This will save the structure of the database, but not the data (which may be huge). For information on how to do this, see the *Local InterBase Server User's Guide* or the online help provided with the Server Manager.
2. Restore the database to the Local InterBase Server. This will be the development platform.
3. Populate the database with a modest amount of representative data. Your application will access this data during the development and testing process. Because it is "dummy" data that is not connected to the production database, there is no chance that the production database will be corrupted. Your application can access any server features present in the production database, including stored procedures.

Once the application has been sufficiently tested against the development database, configure your dBASE application to use an alias that refers to the production server. It is recommended to first test the application against a duplicate of the production database on the server.

To build an application that will access a database that does not already exist on an InterBase Workgroup Server, use the Local InterBase Server (LIBS) as an intermediate data source. This enables you to address all the SQL development issues separately from connectivity and client/server issues. The steps include:

1. Defining the database on the Local InterBase Server.
2. Developing the application against the LIBS database.
3. Migrating the LIBS database to the target server.
4. Redirecting the application to access the target server.