Migration from MS-SQL to Firebird

Marcelo Lopez Ruiz

Public Domain notice and disclaimer

The author has placed this work in the Public Domain, thereby relinquishing all copyrights. Everyone is free to use, modfy, republish, sell or give away this work without prior consent from anybody.

This documentation is provided on an “as is” basis, without warranty of any kind. Use at your own risk! Under no circumstances shall the author(s) or contributor(s) be liable for damages resulting directly or indirectly from the use or non-use of this documentation.

2003

Table of Contents

Microsoft SQL Server (MS SQL) is a widely used database server. There are three versions which currently account for the majority of the user base: MS SQL 6.5, MS SQL 7 and MS SQL 2000.

Introduction

This section describes the conversion guide itself.

The conversion from MS SQL server documentation is meant, first, to help users evaluate whether the process should be performed at all. It then goes on to detail how this can be done, adding bits of experience collected by different people.

There are two important things to take into account when migrating. First, moving the data from one database server to another can be trivial or not, depending on your database schema. There are many tools to help you with this process. The standard data conversions are listed in this document.

Second, you will need to migrate any stored procedures and triggers manually. This is the tricky part. There are many differences, some minor, some important; this document attempts to address most of them, giving examples on the most frequent problems and how to solve them.

Pros and Cons

This section describes reasons to migrate an existing database to Firebird, and reasons not to.

Why migrate to Firebird

This depends mostly on what version you are currently using and what you are using MS SQL for.

For example, if using MS SQL 6.5, it is a simple matter of considering the features and ease of use. MS SQL 6.5 will work with fixed devices rather than dynamically expanding files, which makes it very difficult to balance ease of administration vrs. available space. There are numerous bugs and annoying behaviours which

If you are using MS SQL 7, you know a lot of the little quirks have been removed, but you are still missing some great features, such as updateable views, greater control over identity fields, user-defined functions, and selectable stored procedures. You also don't get cascading referential integrity until the 2000 version. Ditto for using different collation orders in the same database.

For UNIX-like environment, Firebird can have its security integrated with the operating system's. However, this should be discouraged for portability.

MS SQL 2000 improved on MS SQL 7, but is still missing one of the key pieces of Firebird: the multi-generational architecture, which enabled long-running queries to work without getting in the way of traditional short-lived operational transactions. MS SQL will instead try to convince users to buy yet another server (hardware, operating system and database server), set it up as a data warehouse, and use this second server as a source for reports. One can only wonder about the need for a data warehouse in an integrated environment with cross-database query capabilities.

Another reason to migrate is to avoid vendor lock-in. MS SQL will only run on Windows NT/2000 (there are so-called personal editions, but these are limited in available connections and features). This

means you are tied to Microsoft for your operating system and your database server. Firebird will run on many platforms, including Microsoft Windows, Linux, Solaris, MacOS X, and others.

Yet another reason to consider is price. Firebird is free; MS SQL will require a considerable amount of money on a per-processor basis. For example, a database accessed through the Internet on a dual-Pentium machine will cost $39,998 (prices obtained from Microsoft's site on 2001.05.03).

Last, but certainly not least, is the fact that Firebird is open source. This not only means that there are hundreds of developers willing to help you use it, improve on it, find bugs, etc., but that you can even modify it and rebuild it yourself to "scratch your itches". Adding features such as an integrated e-mail system or logging is a matter of understanding the source code and having the available expertise to modify it. While this may not be a trivial task, it is certainly doable, and brings an enormous degree of flexibility.

Why not migrate to Firebird

The first, overriding reason should be because your system is working fine as it is. If this is the case, consider Firebird for future projects, but do not break what is currently working.

There are a number of features MS SQL 7 has that you will not find in Firebird, such as integrated replication support (this is available as an add-on to Firebird), temporary tables, and integration with other database systems through OLE DB. It also has an OLAP Analysis services built into it, and native full-text search (this is available as an add-on to Firebird).

Backups and restores can be done incrementally on MS SQL; Firebird will only backup and restore whole databases at a time.

On Microsoft environments, MS SQL 7 and above can have its security integrated with the operating system's. However, this should be discouraged for portability and performance issues.

MS SQL 2000 also has the ability to work with XML directly, and supports partitioned views for better performance on tables which span several servers.

In general, it would seem that MS SQL has better performance on Windows than Firebird on Windows does. It also has better integration with Microsoft Visual Studio.

Database Server Setup

This section describes differences when installing the database server software.

For users who have used MS SQL in the past and are new to Firebird, they will be greatly surprised when they learn how easy it is to set up Firebird. The setup process is straightforward, and you can connect to your database server immediately after setup. Note that the default system administration username is SYSDBA and the password is masterkey; in MS SQL, the username is sa and the password is empty.

Note that you don't have to select the collation and character sets when installing a Firebird server. In MS SQL, you not only have to select this option up front, but must reinstall to change it; in addition to this, many other software packages, such as Microsoft's own Commerce Server, will refuse to use the server if you select the wrong choices.

For users coming from MS SQL 6.5, Firebird has no notion of devices. All data are kept in files in the normal file system available. Note that you cannot use a raw disk partition to hold your databases.

Important Note: MS SQL uses a logging mechanism to keep database consistent and survive crashes. Firebird uses a multi-generation mechanism to create copies in-place as they are required, but these are not written immediately to disk. While this provides a considerable speed gain, you can turn Forced Writes on a per-database basis to ensure that sudden blackouts will not compromise data integrity. If your server has a reliable environment such as a dedicated Linux box, and some form of UPS, turning Forced Writes on can be ignored.

Database Administration

This section describes differences in how database are managed in Firebird and MS SQL.

Database Files Administration

MS SQL 6.5 uses devices, which can be files or raw partitions, to manage data. This resulted in a hard-to-maintain system. MS SQL 7 and MS SQL 2000 corrected this by using normal files in place of devices. For each database, you will have at least two files: one with the database information itself, and one with a log of transactions performed.

Firebird does not rely on a log to keep track of transactions, and therefore uses a single file to keep everything.

The CREATE DATABASE statement in Firebird is simpler than the CREATE DATABASE statement in MS SQL; see the SQL reference for a full description of its capabilities.

One significant difference between the file management model is that MS SQL uses filegroups to partition a database over a set of files. Firebird can also use different files, but the model is simpler.

An additional consideration for Firebird is the use of shadow files. Shadow files are an instant replica of the database itself. It is typically used to have a hot backup readily available. MS SQL has no such feature, although MS SQL 2000 has a similar capability by the use of log shipping between database servers and replication.

User Administration

In MS SQL 6.5, there are two objects to manage: logins and users. Logins specify a username/password combination used to access a database server; users specify the access rights on each database. Logins are then mapped to users in databases.

In MS SQL 7, a new kind of object is added to manage groups of users: roles. These simply security definition. Some roles are system-defined, such as backup operators or database administrators.

Firebird has a security model similar to MS SQL's, but without logins. Users supply a username, a password, and a role they wish to work under. There is a single security database per database server, which holds all information about permissions on every database, for every user, for every role.

Under both database systems, it is considered good practice to access all resources through stored procedures, and grant access only to stored procedures. Security can then be setup through the security

assigned to stored procedures (in Firebird; in MS SQL, the stored procedure executes using the rights of its creator).

Backup and Restore Operations

Firebird uses a backup and restore model which is much simpler than MS SQL, although it sacrifices flexibility. Backups are performed through command-line or GUI tools, and they backup a whole database at a time. A restore operation will restore a whole database on a server.

There is no operation to backup differences only, or to restore an isolated set of transactions.

Note that there is a very important option when backing up a database in Firebird: platform dependant or portable. Performing a portable backup allows the administrator to backup a database on an operating system and restore that same database on another. This is typically used when development is performed on Windows workstations, and the operational database is then deployed on a more powerful Linux server, for example.

Data Types

This section describes the different data types available in Firebird and MS SQL, and how to translate types from one system to another.

MS SQL has different data types, depending on the version. The following table lists the data types along with the version in which they were introduced.

Table 1. Data Types Conversion Table

MSSQL Ver	Data Type	Firebird	MSSQL definition and comments
6.5	bigint	INT64	8-byte integer type.
6.5	binary	CHAR	Fixed-length binary data with a maximum length of 8,000 bytes. In 6.5, maximum was 255.
6.5	bit	CHAR(1)	Integer data with either a 1 or 0 value. Typically, replaced by constants 'T' and 'F'.
6.5	char	CHAR	Fixed-length non-Unicode character data with a maximum length of 8,000 characters. In 6.5, maximum was 255. Firebird can hold up to 32,767 characters.

MSSQL Ver	Data Type	Firebird	MSSQL definition and comments
6.5	cursor		A reference to a cursor. This can only be used inside stored procedure or triggers; it cannot be used on table declarations.
6.5	datetime	TIMESTAMP	Date and time data from January 1, 1753, to December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.
6.5	decimal	DECIMAL	Fixed precision and scale numeric data from -10^{38} -1 through 10^{38} -1.
6.5	float	FLOAT	Floating precision number data from -1.79E + 308 through 1.79E + 308.
6.5	image	BLOB	Variable-length binary data with a maximum length of 2^{31} - 1 (2,147,483,647) bytes.
6.5	int	INTEGER	Integer (whole number) data from -2^{31} (-2,147,483,648) through 2^{31} - 1 (2,147,483,647).
6.5	money	DECIMAL(18, 4)	Monetary data values from -2^{63} (-922,337,203,685,477.5808) through 2^{63} - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit.
7	nchar	CHAR(x) CHARACTER SET UNICODE_FSS	Fixed-length Unicode data with a maximum length of 4,000 characters.
7	ntext	BLOB SUB_TYPE TEXT	Variable-length Unicode data with a maximum length of 2^{30} - 1 (1,073,741,823) characters.
6.5	numeric	NUMERIC	In MS SQL, decimal and numeric are synonims.
7	nvarchar	VARCHAR(x) CHARACTER SET UNICODE_FSS	Fixed-length Unicode data with a maximum length of 4,000 characters.

MSSQL Ver	Data Type	Firebird	MSSQL definition and comments
6.5	real	DOUBLE	Floating precision number data from -3.40E + 38 through 3.40E + 38.
6.5	smalldatetime	TIMESTAMP	Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute. Firebird's has greater range and accuracy.
6.5	smallint	SMALLINT	Integer data from -2^{15} (-32,768) through $2^{15} - 1$ (32,767).
6.5	smallmoney	DECIMAL(10, 4)	Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit. Note that Firebird's range is greater with this declaration.
2000	sql_variant	BLOB	Allows the storage of data values of different data types.
2000	table	none	Stores results temporarily for later user.
6.5	text	BLOB SUB_TYPE TEXT	Variable-length non-Unicode data with a maximum length of $2^{31} - 1$ (2,147,483,647) characters.
6.5	timestamp	INTEGER	A database-wide unique number. In Firebird, you will have to manage uniqueness through generators.
6.5	tinyint	SMALLINT	Integer data from 0 through 255. Firebird does not have such a small data type.
6.5	varbinary	CHAR	Variable-length binary data with a maximum length of 8,000 bytes.
6.5	varchar	VARCHAR	Variable-length non-Unicode data with a maximum of 8,000 characters. Firebird can hold up to 32,765 characters. In 6.5, maximum was 255.
7	uniqueidentifier	CHAR(38)	A globally unique identifier (GUID). In Firebird, you will have to generate the values with User-Defined Functions (UDFs).

A subtle difference in the way NUMERIC and DECIMAL behave in Firebird to bear in mind is that the NUMERIC definition means *exactly* the precision requested (total number of digits), while DECIMAL mean *at least* the request precision (the digits to the right of the decimal symbol, however, are maintained exactly). In MS SQL, on the other hand, numeric and decimal are synonims.

There is also a very common quasi-data type, identity, which can only be used when defining tables. This is an int which is automatically assigned a value on insertion and cannot be changed.

Converting the bit data type

The bit data type is used to hold a single boolean value, 0 or 1. MS SQL does not support assigning NULL to this fields. InterBase can emulate this with an INTEGER or a CHAR(1) data type.

The acceptable values can be restricted using domains. For more information on Firebird domains, see the Data Definition documentation.

Converting the identity data type

There are many ways to perform the conversion. In general, Firebird is more flexible and powerful in this respect.

The most direct conversion is to create a BEFORE trigger on the table, assigning to the previous column the value from a generator. This ensures that the number is unique.

For added flexibility, a single generator can be used for many tables. In this case, the type would work in a similar way as a timestamp would - by creating a database-wide unique identifier.

Another common technique is to create a stored procedure to allow access to the generator, and allow clients to pre-fetch the number. This is particularly useful for tools such as Delphi which import the NOT NULL constraint on primary keys and refuse to post records with NULL values.

```
CREATE TABLE my_table (
  my_number integer not null primary key
)

CREATE GENERATOR my_generator

CREATE TRIGGER my_before_trigger FOR my_table
BEFORE INSERT
AS
BEGIN
  IF (NEW.my_number IS NULL)
    THEN NEW.my_number = GEN_ID(my_generator, 1);
END

CREATE PROCEDURE get_my_generator
RETURNS (new_value INTEGER)
AS
BEGIN
  new_value = GEN_ID(my_generator, 1);
END
```

Converting the uniqueidentifier data type

MS SQL depends on uniqueidentifier data types for replication. It is also a handy way of creating a world-wide unique identifier for a record.

To use the field like this, create a BEFORE trigger on the table with the field, and retrieve the value from a UDF.

TODO: write the UDF and write the importing procedure

SQL Syntax

This section describes differences in the SQL syntax used by Firebird and MS SQL in general.

Firebird and MS SQL can both use object names (table names, field names, etc.) directly, when they have no whitespace or other symbols. To include whitespace and otherwise escape object names, MS SQL uses brackets, [and], while Firebird uses double quotes, ". Another thing to bear in mind is that MS SQL accepts a database.username.objectname syntax to name objects, which Firebird does not.

> **Warning**
>
> Bear in mind that MS SQL is case-sensitive in its object naming if it was installed with the case-sensitive option; otherwise it's case insensitive. Fun. Not.

> **Tip**
>
> MS SQL also accepts quoted identifiers, but by default it is set only when accessed through OLE DB and ODBC, and not when accessed through the DB-Library. In general, therefore, this practice is discouraged.

MS SQL 7 and above supports modification on joins (update, delete, insert). Firebird has no such syntax.

Data types are, of course, different for the different database. Both support a common subset with the most-used types; this is rarely an issue.

There are different built-in functions. Most of MS SQL functions can be replaced and extended by UDFs in Firebird.

There are different formats for specifying date constants. In general, Firebird will accept different formats independently of the underlying platform - MS SQL, on the other hand, uses a mixture of server-independent, server-side platform and per-client-connection formats. In addition to this, the MS SQL access methods typically introduce one or two additional layers where a string constant may be changed one way or another into a date.

MS SQL has more environment variables than Firebird does, but the most common ones (identity retrieval and user name retrieval) can be found. The only important variable missing is the row count of the latest operation.

An important difference is that Firebird does not support the MS SQL CASE statement. You can sometimes use a stored procedure in its stead, which promotes reusability and eases maintenance.

A minor difference is that MS SQL does not use a delimiter between statement. This can be the source of some tricky bugs, specially when using many parenthesis. Firebird requires that every statement end in a semicolon ; so errors are easier to spot.

Both MS SQL and Firebird support comments using the /* and */ delimiters. MS SQL also supports the -- syntax to comment a single line. Some client-side Firebird tools also support this syntax, but it is not supported in-line.

Using Database Basics

MS SQL allows clients to use many databases from a single connection. To do this, you can use the dbname.user.syntax, or execute an explicit USE statement.

Firebird does not allow you to use different databases in the same SQL statement, but it does allow you to perform transactions spanning multiple databases.

There are many generic tools to enter SQL commands to your database, in both platforms. Note that for Firebird, you do not need to use **GO** to delimit T-SQLbatches; rather, you manage transactions explicitly. You can also use the default of commit-every-statement on both servers.

> **Warning**
>
> If you MS SQL and Firebird setup on the same computer, be careful with the **isql** command. If you do not reference them by the full path, the one which is first on your system path will be used, and both MS SQL and Firebird have a command-line **isql** program.

Using variables

Variable handling is similar on both platforms. Variables must be declared before being used, specifying their types. However, bear in mind that variables names in Firebird need not be prefixed with a @ character, and they do need to be declared before the procedure or trigger body.

For example, compare the following code snippets.

```
/* MS-SQL */
CREATE PROCEDURE my_procedure
AS
DECLARE @my_variable int
SET @my_variable = 5

/* Firebird */
CREATE PROCEDURE my_procedure
AS
DECLARE VARIABLE my_variable int;
BEGIN
  my_variable = 5;
END
```

Under both database servers, parameters are considered normal variables, set to an initial value.

Flow Control

Under both database servers, the **BEGIN** and **END** keywords can be used to group multiple statements, for example, inside an **IF** branch.

The one important flow control statement missing in Firebird is the **CASE** statement. This statement can be used as a **switch** statement in C or a **case** statement in Pascal to change one value for another. This can usually be translated to Firebird as a stored procedure returning some value.

```
/* This is the original MS SQL
   statement, using the * traditional pubs database. */
CREATE PROCEDURE list_states
AS
SELECT
  CASE state
    WHEN 'CA' THEN 'California'
    WHEN 'UT' THEN 'Utah'
    ELSE 'unknown'
  END
FROM authors

/* This is how it can be converted to Firebird. */
/* Isolate the CASE statement. */
CREATE PROCEDURE get_state_name ( state_code char(2) )
RETURNS ( state_name varchar(64) )
AS
BEGIN
 IF (state_code = 'CA') THEN state_name = 'California';
 ELSE IF (state_code = 'UT') THEN state_name = 'Utah';
 ELSE state_name = 'unknown';
END

/* This is the selectable stored procedure. */
CREATE PROCEDURE list_states
RETURNS (state varchar(64))
AS
DECLARE VARIABLE short_state CHAR(2);
BEGIN
  FOR SELECT state FROM authors INTO :short_state DO
  BEGIN
    EXECUTE PROCEDURE get_state_name :short_state
    RETURNING_VALUES :state;
    SUSPEND;
  END
END
```

Three things should be noted from the example above. First, the conversion is trivial. Second, it is however quite verbose. Third, using a stored procedure allows for greater flexibility, and makes maintenance easier. Suppose the **CASE** statement for the state occurs in twelve different procedures, and a new state was added; or that you misspelled a state name; or any other change. It is clearly beneficial to abstract this conversion, trivial as it may seem, into its own stored procedure.

Firebird has no **GOTO** statement. However, this usually turns for the better. **GOTO** statements are usually used in MS SQL because errors do not roll back transactions by default (the @@ERROR variable must be examined after every statement); **GOTO** is used to group error handling statements. In Firebird, there is a better error-handling mechanism: the **WHEN...DO** statements.

Of course, **GOTO** statements can be used for other purposes. In these cases, using stored procedures correctly will usually improve the database design.

The **IF..ELSE** statement exists on Firebird with the same semantics. However, Firebird syntax requires a THEN after the IF condition clause.

```
IF (something = 'unknown')
  THEN something = 'uuhhh.....';
  ELSE something = 'I know! I know!';
```

The **RETURN** statement in MS SQL will return an output integer variable and stop execution. Firebird supports the **EXIT** statement, which will jump to the final **END** in stored procedures. However, there is no implicit output variable, so if you need to return a code (it's optional in MS SQL), you will need to declare an output variable in the procedure.

The **WAITFOR** statement in MS SQL will suspend execution for an amount of time, or until a specified time is reached. Something like this could be done with a UDF; however, under both database servers, an alternative would be very much preferred, as the connection from the client remains suspended, too.

WHILE exists in Firebird as it does in MS SQL, with some differences. There are no **BREAK** or **CONTINUE** statements, but these can be emulated with additional controls and variables. There's also a small difference in syntax; Firebird requires a DO keywords after the **WHILE** condition. Compare the following equivalent snips.

```
/* Firebird syntax. */
WHILE (i < 3) DO
BEGIN
  i = i + 1;
  j = j * 2;
END

/* MS SQL syntax. */
WHILE (i < 3)
BEGIN
  SET @i = @i + 1
  SET @j = @j * 2
END
```

Standard Statements

The standard statements which can be found in all databases are **SELECT**, **INSERT**, **UPDATE** and **DELETE**. Firebird and MS SQL support them, but there are some non-standard MS SQL extension to consider if they are being used.

The SELECT statement in Firebird does not allow the INTO clause to create a new table on the fly. Instead, it is used to bind a result into a variable.

```
/* MS SQL syntax to get field values into a variable. */
SELECT @my_state = state
FROM authors
WHERE auth_name = 'John'

/* Firebird syntax. */
SELECT state INTO :state /* --> note the ":" before the name */
FROM authors
WHERE auth_name = 'John'
```

In MS SQL 7 and above, the **SELECT** clause can take a **TOP** specifier to limit the number of rows returned. This feature is currently under development for the Firebird engine.

Both MS SQL and Firebird support the normal **INSERT** syntax and the **INSERT..SELECT** syntax.

Both MS SQL and Firebird support the normal **UPDATE**. MS SQL also supports a form of **UPDATE** in which a join is performed, and one side of the join is updated. Think of this as a **WHERE** on steroids. If this feature is absolutely required, it can be implemented using views.

Both MS SQL and Firebird support the normal DELETE. MS SQL also supports the TRUNCATE TABLE statement, which is a more efficient (but dangerous) form of DELETE.

```
/* MS SQL syntax to delete all records in my_table. */
TRUNCATE TABLE my_table /* ...or... */
DELETE FROM my_table

/* Firebird syntax. */
DELETE FROM my_table
```

> The biggest threat are our fumbling fingers. More data has been destroyed by "delete from xxx" "oops" than deliberate "delete rdb$pages".
>
> —Jim Starkey

Using Transactions

Transactions are rarely used directly in Firebird when using DSQL. Named transactions are not supported in this case. Both syntaxes accept the **WORK** keyword for compatibility.

This should not present a problem in most situations, as MS SQL's explicit transaction control is usually in place because there no support for using exception handlers.

> **Tip**
>
> MS SQL has a `XACT_ABORT` global variable, to manage whether transactions are rolled back on run-time errors. Otherwise, the `@@ERROR` variable must be examined after each statement.

In general, most discussions about isolation level problems found in MS SQL environments are void when taken to a Firebird database server. Contention between readers and writers is minimal and is resolved by the multigeneration architecture.

Using Cursors

MS SQL uses cursors mostly to iterate over query results to perform activities. Other than syntax, there is little difference in what can be accomplished in either database. Although there are many options for iterating backwards and forwards, in practice the only cursor used is the forward-only cursor.

```
/* MS SQL syntax. */
DECLARE my_cursor CURSOR
FOR SELECT au_lname FROM authors ORDER BY au_lname
  DECLARE @au_lname varchar(40)
  OPEN my_cursor
  FETCH NEXT FROM my_cursor INTO @au_lname
  WHILE @@FETCH_STATUS = 0
  BEGIN
    /* Do something interesting with @au_lname. */
    FETCH NEXT FROM my_cursor
  END
CLOSE my_cursor
DEALLOCATE my_cursor
```

```
/* Firebird syntax. */
DECLARE VARIABLE au_lname VARCHAR(40);
  ...
  FOR SELECT au_lname FROM authors
  ORDER BY au_lname INTO :au_lname DO
  BEGIN
    /* Do something interesting with au_lname. */
  END
```

Note that MS SQL can place cursors in variables and pass them around; this cannot be performed in Firebird.

> **Warning**
>
> Different versions of MS SQL change the default scope for cursor variables. Be careful with how you use them and bear this in mind when reading code to convert it.

Server-Side SQL

This section goes beyond simple SQL syntax differences, and describes the different tasks which can be accomplished server-side through SQL.

SQL Tricks

This section shows how to use some advanced Firebird features to emulate MS SQL behaviour, and avoid MS SQL workarounds.

Trick: Using Cascades

Versions of MS SQL previous to 2000 will not support cascading updates and deletes. Foreign keys will always roll back on changes.

In MS SQL, this is typically solved by a combination of stored procedures and triggers, to avoid declaring the foreign key explicitly. This, in turns, makes the relationships implicit rather than explicit, which means that tools can't read the relationships directly and work on them.

During the migration process, all of the workaround procedures and triggers can be ignored - Firebird supports the cascading updates and deletes enforcing declaratively.

Trick: Using Updateable Views

Versions of MS SQL previous to 2000 will not support updates on joined views fully. This is a major issue, since views are typically considered read-only; there are numerous restrictions to have them being updateable.

In Firebird, there are also a number of restrictions, but they are only meant for *automatic* updating. If the engine cannot perform the updates by itself, you can write triggers on the view to perform the required logic.

Client Access

This section describes the differences in how clients access a Firebird and an MS SQL database.

Built-in Client Access Tools

The standard command-line utility is **isql**. This is used usually when executing large scripts, or when writing batch files.

When a graphical user interface (GUI) is available, the administration tool will most probably be IB-Console. This tool is similar to MS SQL's Enterprise Manager.

Client Access for Developers

There are three basic mechanisms to get to a Firebird database. You can either use the raw C API interface, an Open Database Connectivity (ODBC) driver, or an OLE DB driver (the latter is used also for ActiveX Data Objects, ADO, access).

Using the raw C API allows developers to write portable code. All platforms support this API. This is also the foundation for the popular Delphi and C++ Builder component sets, such as InterBase Objects (IBO) and InterBase Express (IBX).

Using an ODBC driver lets developers write code that can be reused with different databases, as long as they restrict themselves to a common SQL subset. There are many tools which can use Firebird through ODBC drivers.

Using an OLE DB drivers lets developers use Microsoft's popular ADO API. This allows the Firebird database to be reached from tools such as Visual Basic or ActiveX Server Pages (ASP). The most popular driver is currently Microsoft's own OLE DB->ODBC bridge.

Tools

This section describes tools used to aid in migration, and to replace standard tools found in MS SQL.

Migration Tools

The following table lists tools which can be used to help you migrate an MS SQL database to a Firebird database.

Table 2. Migration Tools

Tool Name	Provided By
Microsoft Access and Microsoft SQL Server to InterBase Wizard	Marcelo Lopez Ruiz
IBDataPump	Alex Poloziouk

Replacing MS SQL Tools

The following table lists tools which can be used to replace the standard tools that come with MS SQL.

Note that, in general, all applications that provide services similar to Enterprise Manager also provide query services.

Table 3. Replacement Tools

Tool Name	Replaced By
Books Online	Online PDF Documentation
Client Network Utility	No need to; connection strings are fully self-described. Test with IBConsole.
Enterprise Manager	IBConsole
Enterprise Manager	IBExpert (commercial)
Enterprise Manager	InterBase Workbench (commercial)
Import and Export Data	-
Performance Monitor	InterBase Heartbeat (commercial)
Profiler	InterBase Observer SQL Profiler (commercial)
Query Analyzer	IBConsole

Tool Name	Replaced By
Server Network Utility	IBConsole
Service Manager	Control Panel Applet

These tools are just samples to help in replacement. For a better and more up-to-date list, see IB-Phoenix Contributed Downloads