Firebird Interactive SQL Utility

Kamala Vadlamani, Paul Vinkenoog, Norman Dunbar

Version 0.6, 20 June 2020

Table of Contents

Chapter 1. Introduction

This manual provides reference material for the Firebird *Interactive SQL Utility* (`isql`), and instructions on how to use it to perform tasks within the database.

This manual is a work in progress. It is subject to change and possible restructuring as new versions appear.

1.1. Audience

This manual assumes prior knowledge of basic database concepts.

1.2. Version

This manual describes the `isql` utility in Firebird version 1.5 and higher.

Chapter 2. Overview

The `isql` utility is a text-mode client tool located in the `bin` directory of the Firebird installation. It provides a command line interface for interactive access to a Firebird database. It accepts DSQL statements along with a group of `SET` and `SHOW` commands to query and interact with the Firebird database. Some `SET` commands can be incorporated in DDL scripts to perform batch executions within `isql`. It also accepts DDL, DML and console commands.

The `isql` utility can be used in three modes: as an interactive session; directly from the command line; and as a non-interactive session, using a shell script or batch file. Different tasks may be performed in each of the modes, as illustrated below:

- An interactive session can be invoked from the command line of the operating system shell, and lasts until the session is terminated, using a QUIT or EXIT command. `Isql` can be used interactively to:
  - Create, update, query, and drop data or meta data.
  - Input a script file containing a batch of SQL statements in sequence without prompting.
  - Add and modify data.
  - Grant user permissions.
  - Perform database administrative functions.
- Directly from the command line, with individual options and without starting an interactive session. Commands execute, and upon completion, return control automatically to the operating system.
- In a non-interactive session, the user employs a shell script or batch file to perform database functions.

Because other applications in the Linux environment, for example, MySQL, also use `isql` as a utility name, you are advised to run the Firebird utility from its own directory, or provide the absolute file path if you have another relational database, besides Firebird, installed on your machine.

Some of the affected distributions, such as Mint Linux — based on Ubuntu — have renamed Firebird's `isql` to be `isql-fb`. There may be similar changes in other distributions.

In addition, not all distributions install Firebird to the same location. OpenSuse, for example, installs just about everything to `/opt/firebird/n.n` but Ubuntu and derivatives install it to a number of different locations but the utilities are in `/usr/bin`.

For the remainder of this document, the assumption will be that the utility is called `isql`.

Chapter 3. Invoking Isql

If you do not have the Firebird bin directory on your path, then either go to the bin subdirectory of your Firebird installation and type isql (Windows) or ./isql (Linux) at the command prompt, or, type the full path to the isql application to execute it. If the bin is on your path, you may start it by typing isql regardless of your operating system.

Example:

```
C:\Firebird_2_0\bin>isql

Use CONNECT or CREATE DATABASE to specify a database
SQL> CONNECT "C:\DATABASES\FIREBIRD\MY_EMPLOYEE.FDB"
CON> user 'SYSDBA' password 'secret';
```

The above is the simplest method of starting isql, and once activated in this way, you must begin by either creating a new database, or connecting to one. The prompt given by isql is a hint as to what you must do next. If you wish to connect to an already existing database, you may pass the database name on the command line. You should be aware that unless you also pass the username and password as well, you may see an error message telling you that your username or password have not bee defined. In this case, you need to supply the username and password, or create two environment variables as discussed below.

The following example shows how to pass the database name plus user credentials on the command line.

```
C:\Firebird_2_0\bin>isql -user sysdba -password secret employee

Database:  employee, User: sysdba

SQL>
```

In this example, we used a database alias for the employee database. This example comes predefined in the file aliases.conf which normally lives under the directory that Firebird was installed in, but some Linux distributions put it in /etc/firebird/n.n where *n.n* is the version of the Firebird database server. There is more information on connecting to databases, using full paths or alias names, below.

Chapter 4. Command Line Switches

Command line switches are arguments that begin with a minus/hyphen ('-') character. The following is an example of what happens when you attempt to start isql with an invalid switch — it displays the list of valid switches with a brief explanation of each.

```
tux> $ isql --help

Unknown switch: -help
usage:    isql [options] [<database>]
        -a(ll)                  extract metadata incl. legacy non-SQL tables
        -b(ail)                 bail on errors (set bail on)
        -c(ache) <num>          number of cache buffers
        -ch(arset) <charset>    connection charset (set names)
        -d(atabase) <database>  database name to put in script creation
        -f(etch_password)       fetch password from file
        -e(cho)                 echo commands (set echo on)
        -ex(tract)              extract metadata
        -i(nput) <file>         input file (set input)
        -m(erge)                merge standard error
        -m2                     merge diagnostic
        -n(oautocommit)         no autocommit DDL (set autoddl off)
        -nod(btriggers)         do not run database triggers
        -now(arnings)           do not show warnings
        -o(utput) <file>        output file (set output)
        -pag(elength) <size>    page length
        -p(assword) <password>  connection password
        -q(uiet)                do not show the message "Use CONNECT..."
        -r(ole) <role>          role name
        -r2 <role>              role (uses quoted identifier)
        -s(qldialect) <dialect> SQL dialect (set sql dialect)
        -t(erminator) <term>    command terminator (set term)
        -u(ser) <user>          user name
        -x                      extract metadata
        -z                      show program and server version
```

Not all of these switches appear in every release of Firebird. Some will be seen in more recent releases. Many of the switches have an equivalent set command, and these will be discussed below.

Using `-b(ail)`

The command line switch `-b(ail)` instructs the `isql` utility to bail on error, but only when used in a non-interactive mode. The switch returns an error code to the Operating System.

This switch was added to prevent `isql` from executing scripts after an error has been detected. No further statements will be executed and `isql` will return an error code to the OS.

Users still need to use the `-e(cho)` switch to echo commands to an output file, to isolate the exact statement that caused the error.

When the server provides line and column information, users can see the exact line of the DML in the script that caused the problem. When the server only indicates failure, users can view the first line of the statement that caused the failure, in relation to the entire script.

This feature is also supported in nested scripts. For example, Script A includes Script B and Script B causes a failure, the line number is related to Script B. When Script B is read completely, `isql` continues counting the lines related to Script A, since each file gets a separate line counter. Script A includes Script B when Script A uses the INPUT command to load Script B.

Lines are counted according to what the underlying IO layer considers separate lines. For ports using EDITLINE, a line is what readline() provides in a single call. The line length limit of 32767 bytes remains uncharged.

Using `-ex(tract)`

The command line switch `-ex(tract)` can be used to extract meta data from the database. It can be used in conjunction with the `-o(utput)` switch to extract the information to a specified output file.

The resultant information can be used to view all the changes made to the database since its creation. Before making any more changes, create a new database with identical schema definitions or new ones, or create a new database source file.

Using `-m2` and `-m(erge)`

The command line switch `-m2`, has been added in Firebird 2.0 and can be used to send the statistics and plans to the same output file that receives the input from the `-o(utput)` switch.

In earlier versions of Firebird (before version 2.0), when a user specified that the output should be sent to a file, two options existed: the command line switch `-o(utput)` with a file name to store the output, or the command `OUTput` with a file name to store the output. Both these options could be employed either in a batch session or in the interactive `isql` shell. In both cases, simply passing the command `OUTput` would return the output to the console. While the console displayed error messages, these were not sent to the output file.

The `-m(erge)` command line switch, can be used to incorporate the error messages into the output files.

The `-m2` command line switch ensures that the stats and plan information derived from the `SET STATS`, `SET PLAN` and `SET PLANONLY` commands are also sent to the output file and not just returned to the console.

Neither `-m(erge)` nor `-m2` has an interactive counterpart through a `SET` command. They are for use only as command line `isql` options.

_Using `-r2` and `-r(ole)`

This switch can be used to specify a case-sensitive role name. The default switch for this is `-r(ole)`. Roles provided in the command line are uppercased. With `-r2` they are passed to the engine exactly as typed in the command line.

Using `-o(utput)

The `OUTPUT` switch allows users to store records of commands to a script file. The `TMP` setting on a client can be used to control where these script files will be stored, if an absolute file path is not specified.

Chapter 5. Starting An Isql Session

To begin an isql session, enter the command line options and the name of the database in the Linux /Unix shell or Windows command console. For example:

```
isql [options] [database_name_or_alias]
```

When invoking isql, you will need to include an appropriate -user and -password in your options, unless users have the ISC_USER and ISC_PASSWORD declared as operating system variables. For example:

```
isql -user SYSDBA -password masterkey
```

Isql starts an interactive session if no options are specified. If no database is specified, users must connect to an existing database or create a new one after starting isql. It starts the interactive session by connecting to the named database, provided the login options are accurate and valid for the specified database. Depending on the options specified, isql starts an interactive or non-interactive session.

Reading an input file and writing to an output file are not considered interactive tasks, therefore the -input or -output command line options do not initiate an interactive session. Options used to extract DDL statements, such as -a and -x also only initiate a non-interactive session.

Isql can be run from either a local or remote client:

- When connecting using a local client, you may set the environment variables ISC_USER and ISC_PASSWORD. For more information on these, see below.
- When connecting from a remote client, you will need a valid name and password.

Chapter 6. Ending An Isql Session

There are two ways to exit isql.

- If you wish to roll back all uncommitted work and exit isql type this command at the prompt:

```
SQL> QUIT;
```

- If you wish to commit all your work before exiting isql, then type in the following command:

```
SQL> EXIT;
```

Chapter 7. Getting Help

Isql comes with the HELP command. This gives brief details of most of the commands available — unfortunately, some are missing. The help command also allows you to drill down for further information. To activate the help system, simply type HELP at the prompt, as shown below (from Firebird 2.5):

```
SQL> help;

Frontend commands:
BLOBDUMP <blobid> <file>   -- dump BLOB to a file
BLOBVIEW <blobid>          -- view BLOB in text editor
EDIT     [<filename>]      -- edit SQL script file and execute
EDIT                       -- edit current command buffer and execute
HELP                       -- display this menu
INput    <filename>        -- take input from the named SQL file
OUTput   [<filename>]      -- write output to named file
OUTput                     -- return output to stdout
SET      <option>          -- (Use HELP SET for complete list)
SHELL    <command>         -- execute Operating System command in sub-shell
SHOW     <object> [<name>] -- display system information
    <object> = CHECK, COLLATION, DATABASE, DOMAIN, EXCEPTION, FILTER, FUNCTION,
               GENERATOR, GRANT, INDEX, PROCEDURE, ROLE, SQL DIALECT, SYSTEM,
               TABLE, TRIGGER, VERSION, USERS, VIEW
EXIT                       -- exit and commit changes
QUIT                       -- exit and roll back changes

All commands may be abbreviated to letters in CAPitals
```

Most of these commands have no further levels of detail, while the SET command does. To drill down into those extra levels, proceed as follows:

```
SQL> help set;

Set commands:
    SET                     -- display current SET options
    SET AUTOddl             -- toggle autocommit of DDL statements
    SET BAIL                -- toggle bailing out on errors in non-interactive mode
    SET BLOB [ALL|<n>]      -- display BLOBS of subtype <n> or ALL
    SET BLOB                -- turn off BLOB display
    SET COUNT               -- toggle count of selected rows on/off
    SET ROWCOUNT [<n>]      -- limit select stmt to <n> rows, zero is no limit
    SET ECHO                -- toggle command echo on/off
    SET HEADING             -- toggle display of query column titles
    SET LIST                -- toggle column or table display format
    SET NAMES <csname>      -- set name of runtime character set
    SET PLAN                -- toggle display of query access plan
    SET PLANONLY            -- toggle display of query plan without executing
    SET SQL DIALECT <n>     -- set sql dialect to <n>
    SET STATs               -- toggle display of performance statistics
    SET TIME                -- toggle display of timestamp with DATE values
    SET TERM <string>       -- change statement terminator string
    SET WIDTH <col> [<n>]   -- set/unset print width to <n> for column <col>

All commands may be abbreviated to letters in CAPitals
```

If you attempt to drill down into any other command, the effect is exactly the same as executing the `HELP` command on its own.

In the output from `HELP SET`, there doesn't appear to be any help on the `SET TRANSACTION` command.

Chapter 8. Connecting To A Database

A sample database named `employee.fdb` is located in the `examples/empbuild` subdirectory of your Firebird installation. Users can use this database to experiment with Firebird. Note that on some POSIX systems, the example database may not be located in the location given above. Each Linux Distribution, for example, may have decided to relocate some files.

```
CONNECT database_name [USER username] [PASSWORD password] [ROLE role_name];
```

If any of the parameters to the connect command contains spaces, you must wrap that parameter in single or double quotes.

If username or password are not supplied, then the current values in the `ISC_USER` and `ISC_PASSWORD` environment variables are used instead. There is no environment variable to preset the required role.

It is possible to connect to a database using `isql` in two ways: locally and remotely.

- To connect locally, on Windows XP, use the `CONNECT` statement with the full file path or an alias (for a local database):

  ```
  SQL> CONNECT "C:\DATABASES\FIREBIRD\MY_EMPLOYEE.FDB"
  ```

  On Linux, a similar example would be:

  ```
  SQL> CONNECT "/databases/firebird/MY_EMPLOYEE.FDB"
  ```

- If connecting remotely (using TCP/IP), use the `CONNECT` statement with the server name and complete file path of the database or, an alias. When using the full path, remember to ensure that the server name is separated from the database path with a colon.

  To connect to a database on a Linux/UNIX server named cosmos:

  ```
  SQL> CONNECT 'cosmos:/usr/firebird/examples/employee.gdb';
  ```

  To connect to a database on a Windows server named cosmos:

  ```
  SQL> CONNECT 'cosmos:C:\DATABASES\FIREBIRD\MY_EMPLOYEE.FDB'
  ```

Firebird is slash agnostic and automatically converts either type of slash to suit the relevant operating system.

8.1. Using Database Alias Names

In the examples above, we have been using the full path to the database file. This has a disadvantage in that all clients will be able to determine exactly where the database is to be found, or, may cause problems when the database has to be moved. To alleviate these problems, database aliases can be used.

Once Firebird has been installed, a file named `aliases.conf` can be found in the main installation folder. By adding an entry to this folder, the full path to the database can be simplified to an alias name. This makes connecting easier, hides the actual database path from inquisitive users and allows the database to be moved around as necessary without having to change all the clients to allow them to connect to the database at the new location.

To create an alias for the database currently known as `/databases/firebird/MY_EMPLOYEE.FDB` on the cosmos Linux server, we need to add the following to the `aliases.conf` file on the cosmos server. By default, this will be in the folder `/opt/firebird`. On Linux, this file is owned by the root user and so, must be updated by the root user. On Windows, you need to be either an administrator, a power user or SYSTEM to change the file.

```
my_employee = /databases/firebird/MY_EMPLOYEE.FDB
```

There should be no quotes around the path to the database file.

Regardless of where the database file is currently located, or if it has its physical filename renamed, etc, all the local users will refer to the database simply as *my_employee*. Remote users will refer to this database as *cosmos:my_employee*. The following example shows an `isql` session being connected locally to the database using the alias name rather than a full path:

```
cosmos> /opt/firebird/bin/isql my_employee
Database:  test, User: sysdba

SQL>
```

Alternatively, a remote connection would be made as follows, specifying the server name and the database alias together:

```
C:\Program Files\Firebird\Firebird_2_0\bin>isql cosmos:my_employee
Database:  cosmos:my_employee

SQL>
```

Because the alias is defined on the server where the database resides, the remote client needs to supply the server name and the alias name (as defined on that server) in order to make a connection.

Using the `CONNECT` command in an existing `isql` session is equally as simple using alias names:

```
SQL> CONNECT 'cosmos:my_employee';
Database:  cosmos:my_employee

SQL>
```

Regarding the security aspect of using database alias names to hide the full path to the actual database file(s), it's not really all that secure as the following SQL command shows:

```
SQL> select MON$DATABASE_NAME from mon$database;

MON$DATABASE_NAME
================================
/data/databases/firebird/test.fdb
```

Chapter 9. Creating A Database

To create a database interactively using the `isql` command shell, get to a command prompt in Firebird's `bin` subdirectory and type `isql` (Windows) or `./isql` (Linux):

```
C:\Program Files\Firebird\Firebird_2_0\bin>isql
Use CONNECT or CREATE DATABASE to specify a database
```

To create a database named `monkey.fdb` and store it in a directory named `test` on your `C:`-drive:

```
SQL>CREATE DATABASE 'C:\test\monkey.fdb' page_size 8192
CON>user 'SYSDBA' password 'masterkey';
```

In the `CREATE DATABASE` statement it is *mandatory* to place quote characters (single or double) around path, user name and password.

When running Classic Server on Linux, if the database is not started with a host name, the database file will be created with the Linux login name as the owner. This may cause access rights to others who may want to connect at a later stage. By prepending the `localhost:` to the path, the server process, with Firebird 2.0 running as user `firebird`, will create and own the file.

To test the newly created database type:

```
SQL>SELECT RDB$RELATION_ID FROM RDB$DATABASE;

RDB$RELATION_ID
===============
128

SQL> commit;
```

To get back to the command prompt type `quit` or `exit`.

The above technique, as demonstrated, works, but ideally databases and meta data objects should be created and maintained using data definition scripts.

Chapter 10. Setting The ISC_USER And ISC_PASSWORD Environment Variables

An environment variable is a named object that contains information used by one or more applications. They are global to their specific Operating Systems. The Firebird server recognises and uses certain environment variables configured in Windows, Linux and other Unix systems.

The ISC_USER and ISC_PASSWORD environment variables in Firebird are designed to give SYSDBA access to the database from the command line utilities and client applications to anyone who has access to a host machine.

When running command line utilities like isql, gbak, gstat, and gfix, Firebird will search to see if the ISC_USER and ISC_PASSWORD environment variables are set. If you do not provide a user name and password while connecting to a database locally, Firebird will let you log in provided it finds these variables.

For security reasons, it is not advised to specify the SYSDBA user name and password using these two environment variables especially on an insecure computer.

The ISC_USER and ISC_PASSWORD environment variables may be set in order to start isql locally. To set the environment variables:

- In Windows 2000 / XP, this is done in the Control Panel → System → Advanced → Environment Variables. Any changes made here will be permanent. You may also define these variables in a command window prior to running any of the Firebird utilities, such as isql. For example:

```
C:\> set ISC_USER=sysdba
C:\> set ISC_PASSWORD=secret
C:\> isql my_employee

SQL>
```

- In Linux and Unix platforms, this depends on the type of shell being used and how the desktop is configured. Please refer to your Operating System documentation to set environmental variables. For the bash shell, the following example shows the process:

```
cosmos> export ISC_USER=sysdba
cosmos> export ISC_PASSWORD=secret
cosmos> /opt/firebird/bin/isql my_employee

SQL>
```

Chapter 11. SQL Dialects

Firebird supports three SQL dialects in each client and database server. These SQL dialects are differentiated in the context of the date-time format and the precision of a numerical data type. The dialects serve to instruct the Firebird server on how to process features implemented in legacy Borland Interbase databases, earlier than version 6.0. Dialects are set up at runtime and can be changed for the client at connection time or with a `SET SQL DIALECT` command.

Dialect 2 is only used when converting a dialect 1 database to a dialect 3 database.

The following table illustrates the differences between the dialects.

Table 1. SQL Dialects

SQL	Dialect 1	Dialect 2	Dialect 3
Date	Date & Time (Timestamp)	ERROR Message	Date only
Time Stamp	Timestamp (v.6.x only)	Timestamp	Timestamp
Time	Error message	Error message	Time only
<"quoted item">	String	Error message	Symbol only
Precision: 1/3 =	0.3333333... (double precision)	0	0
Numeric 11	double precision	64 bit int	64 bit int

Currently it is possible to create databases in Dialect 1 and 3 only, however it is recommended that you use Dialect 3 exclusively, since Dialect 1 will eventually be deprecated. Dialect 2 cannot be used to create a database since it only serves to convert Dialect 1 to Dialect 3.

When connecting to a database using `isql`, the utility takes on the dialect of the database, unless you specify otherwise. Dialects cannot be set as a parameter of a `CREATE DATABASE` statement. So, when creating a database using `isql`, the database will be in the dialect that is current in `isql` at the time the `CREATE DATABASE` statement is issued. You may set the dialect using the `isql` utility in two ways:

- When you start `isql` type:

```
cosmos> isql -sql_dialect n
```

  (where *n* refers to the dialect number)`

- Within a SQL script or `isql` session, type:

```
SQL> SET SQL DIALECT n;
```

Prior to Firebird 2.0 when `isql` disconnected from a database, either by dropping it or by trying to connect to a non-existent database, it remembered the SQL dialect of the previous connection, which lead to some inappropriate warning messages. This has been fixed in 2.0

Chapter 12. Terminator Character

The default terminator symbol for the Firebird database is the semicolon (';'). Statements will only be executed if they end with a semicolon. However, you may use `isql` to change the symbol to any printable character, or characters, from the first 127 characters of the ASCII subset, by using the `SET TERM` command.

The default terminator maybe changed in all instances except in the case of *Procedural SQL* or PSQL. PSQL does not accept any terminator other than a semicolon.

To change the terminator character to a tilde ('~') enter the following code:

```
SQL> SET TERM ~ ;
```

You must terminate this command with the current terminator of course! Changing the terminator is useful if you wish to type in a PSQL function as the following example shows. Because PSQL will *only* accept the semicolon as a terminator, then `isql` needs to know which semicolon is being used for the PSQL code and which is being used to terminate the SQL commands being entered.

```
SQL> set term ~ ;

SQL> create procedure test_proc (iInput integer = 666)
CON> returns (oOutput integer)
CON> as
CON> begin
CON>   oOutput = iInput;
CON>   suspend;
CON> end~

SQL> set term ; ~

SQL> commit;

SQL> select * from test_proc;

     OOUTPUT
============
         666
```

You can see that within the code for the procedure itself, the terminator is the semicolon. However, outside of the actual procedure code, the terminator is the tilde ('~'). `isql` is processing a single `CREATE PROCEDURE` command, but within that one SQL statement, there are multiple embedded PSQL statements:

```
oOutput = iInput;
suspend;
```

These have the semicolon terminator, as required by PSQL. The end of the `CREATE PROCEDURE` command is indicated by the use of the tilde as the terminator:

```
end~
```

You can, if desired, simply change the terminator because you prefer something other than a semi-colon. You don't have to be writing procedures in order to change it.

```
SQL> -- Change terminator from ; to +
SQL> set term + ;

SQL> select count(*) from employee+

       COUNT
============
          42

SQL> -- Change terminator from + to 'fred'
SQL> set term fred +

SQL> select count(*) from employee fred

       COUNT
============
          42

SQL> -- Change back from 'fred' to ;
SQL> set term ; fred
```

However, you must be careful not to pick a terminator character that will cause SQL statements to fail due to the terminator being used at some point within the SQL statement.

```
SQL> select 600+60+6 as The_Beast from rdb$database;

            THE_BEAST
====================
                  666


SQL> set term + ;
SQL> select 600+60+6 as The_Beast from rdb$database+

Statement failed, SQLSTATE = 42000
Dynamic SQL Error
-SQL error code = -104
-Unexpected end of command - line 1, column 8
...


SQL> set term ; +
```

The presence of the terminator within an expression has caused the "unexpected end of command" error. The SQL Parser within the Firebird database engine has determined that "select 600" is not a valid statement. For this reason, it is best to always choose a character, or characters, that will not confuse the parser.

```
SQL> set term ++ ;

SQL> select 600+60+6 as The_Beast from rdb$database++

            THE_BEAST
====================
                  666
```

Chapter 13. Isql Prompts

The SQL prompt

As shown above, the normal `isql` prompt for input is the `SQL>` prompt. This indicates that the previous command has been completed and `isql` is now waiting for a new command to process.

The CON prompt

The `CON>` or *Continuation* prompt is displayed if users press `Enter` without ending a SQL statement with a terminator. For example:

```
SQL> HELP
CON>
```

Whenever you see the `CON>` prompt, you may either continue entering the remainder of the command, or, enter a terminator to terminate the command. When you press `Enter`, the command will be executed in the latter case.

Chapter 14. Error Handling And Exception Support

Exception handling is a programming construct designed to handle an occurrence that disrupts the normal execution of a program. These are called errors. Exceptions are user-defined named error messages, written specifically for a database and stored in that database for use in stored procedures and triggers.

For example, if it is ascertained in a trigger that the value in a table is incorrect, the exception is fired. This leads to a rollback of the total transaction that the client application is attempting to commit. Exceptions can be interleaved, and shared among the different modules of an application, and even among different applications sharing a database. They provide a simple way to standardize the handling of preprogrammed input errors.

Exceptions are database objects, like Tables, Views and Domains, and are part of the database's metadata. They can be created, modified and dropped like all other Firebird objects using `isql`.

In `isql`, error messages comprise the `SQLCODE` variable and the Firebird status array. The following table provides some examples:

Table 2. ISQL Error Codes and Messages

SQLCODE	Message	Meaning
<0	SQLERROR	Error occurred: statement did not execute
0	SUCCESS	Successful execution
+1 to +99	SQLWARNING	System warning or information message
+100	NOT FOUND	No qualifying rows found, or end of current active set of rows reached

Chapter 15. Transaction Handling

The Firebird architecture allows high transaction concurrency. Transaction save points (nested transactions) are also supported. All Firebird transactions are ACID compliant. ACID is explained below:

Atomicity

ensures that transactions either complete in their entirety or not at all, even if the system fails halfway through the process.

Consistency

ensures that only valid data will be written to the database. If a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules. If a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules, without necessarily preserving consistency at all intermediate levels.

Isolation

ensures that transactions are isolated from one another, even if several transactions are running concurrently. Concurrency refers to a state within the database where two or more tasks are running simultaneously. This way, a transaction's updates are concealed from the rest until that transaction commits. Transactions in Firebird are isolated within separate contexts defined by client applications passing transaction parameters.

Durability

ensures that once a transaction commits, its updates survive within the database, even if there is a subsequent system crash.

There are several parameters available to configure transactions in order to ensure consistency within the database. These parameters invoke the concept of concurrency. To ensure data integrity, there are four configurable parameters affecting concurrency: isolation level; lock resolution mode; access mode; and table reservation.

- *Isolation Level:* A transaction isolation level defines the interaction and visibility of work performed by simultaneously running transactions. There are four transaction isolation levels according to the SQL standard:

  READ COMMITTED

  A transaction sees only data committed before the statement has been executed.

  READ UNCOMMITTED

  A transaction sees changes done by uncommitted transactions.

  REPEATABLE READ

  A transaction sees during its lifetime only data committed before the transaction has been started.

SERIALIZABLE

This is the strictest isolation level, which enforces transaction serialization. Data accessed in the context of a serializable transaction cannot be accessed by any other transaction.

In `isql`, a transaction is begun as soon as the utility is started. The transaction is begun in `SNAPSHOT` isolation, with a lock resolution set to `WAIT`. Since the Firebird `isql` utility accepts DDL, DML and other commands, transactions are handled accordingly, in the following ways:

- DDL statements are committed automatically when issued at the SQL prompt in two ways:
  - When `COMMIT` statements are included in the script.
  - By ensuring the automatic commit of DDL in a `isql` script, by issuing a `SET AUTODDL ON` statement. To turn it off, issue a `SET AUTODDL OFF` statement at the `isql` prompt.
- DML statements are not committed automatically. You must issue a `COMMIT` statement to commit any DML changes to the database.
- You can use various `SHOW` commands in `isql` to query database metadata. Metadata is stored in system tables. When a `SHOW` command is issued it operates in a separate transaction from user statements. They run as `READ COMMITTED` background statements and acknowledge all metadata changes immediately.

Users can specify the access mode and level of isolation for the next transaction, and explicitly commit the current transaction by using the `SET TRANSACTION` statement. *SET TRANSACTION* can be executed only when there is no other transaction being processed. It does not by itself initiate a transaction. Here is the syntax:

```
SQL> SET TRANSACTION;
```

In Firebird 2.0 the `SET TRANSACTION` statement has been enhanced to support all Transaction Parameter Buffer (TPB) options. These include:

- `NO AUTO UNDONE`
- `IGNORE LIMBO`
- `LOCK TIMEOUT number`

Example:

```
SET TRANSACTION WAIT SNAPSHOT NO AUTO UNDONE LOCK TIMEOUT 10;
```

If you request help on the `set` in `isql` then the `set transaction` command is not shown.

Chapter 16. Script Handling

A batch of DDL and/or DML statements in a text file is known as a script. Scripts can be used to create and alter database objects. These are referred to as *Data Definition Language* (DDL) scripts. Scripts that manipulate data by selecting, inserting, updating, deleting or performing data conversions, are called *Data Manipulation Language* (DML) scripts.

One of the most important tasks handled by `isql` is to process scripts. It can handle both DDL and DML Scripts, but they should be included in separate scripts to avoid data integrity problems. This script processing feature of `isql` allows the linking of one script to another using the `isql` command `INPUT <filespec>`. Scripts statements are executed in order that they appear in the script file. The default setting in `isql` for `AUTODDL` is set to `ON`. You may use the `SET AUTODDL` command to control where or when statements will be committed.

The `AUTODDL` setting *only* affects DDL statements. It doesn't commit DML statements. If you mix DDL and DML statements within the same interactive session, then the `AUTODDL` commits *do not* commit your DML transactions. For example:

```
SQL> set autoddl on;

SQL> insert into test(a) values (666);
SQL> commit;

SQL> select * from test;

           A
============
         666

SQL> insert into test(a) values (999);
SQL> select * from test;

           A
============
         666
         999

SQL> create table another_test(b integer);
SQL> rollback;

SQL> select * from test;

           A
============
         666
```

Scripts can redirect their output to a log file using the `OUTPUT file_name` command. This can be entered directly at the `isql` prompt, or as part of a script file itself.

Chapter 17. Isql Commands

Isql commands affect the running of isql itself and do not affect the database or data in any way. These commands are used to display help, run scripts, create listings and so on. You can easily see a list of the available commands by typing the help command which will produce the following output:

```
SQL> help;

Frontend commands:
BLOBDUMP <blobid> <file>   -- dump BLOB to a file
BLOBVIEW <blobid>          -- view BLOB in text editor
EDIT     [<filename>]      -- edit SQL script file and execute
EDIT                       -- edit current command buffer and execute
HELP                       -- display this menu
INput    <filename>        -- take input from the named SQL file
OUTput   [<filename>]      -- write output to named file
OUTput                     -- return output to stdout
SET      <option>          -- (Use HELP SET for complete list)
SHELL    <command>         -- execute Operating System command in sub-shell
SHOW     <object> [<name>] -- display system information
    <object> = CHECK, COLLATION, DATABASE, DOMAIN, EXCEPTION, FILTER, FUNCTION,
               GENERATOR, GRANT, INDEX, PROCEDURE, ROLE, SQL DIALECT, SYSTEM,
               TABLE, TRIGGER, VERSION, USERS, VIEW
EXIT                       -- exit and commit changes
QUIT                       -- exit and roll back changes

All commands may be abbreviated to letters in CAPitals
```

Each of these commands will now be discussed. Note the last line of output from the help command. It explains that each of the commands may be abbreviated to just those letters displayed in capital letters. In the following discussion, the optional characters will be displays, as above, in lower case letters. For example, the input command will be shown as INput to indicate that the characters 'put' are optional.

17.1. Blobdump

```
SQL> BLOBDUMP blob_id filename;
```

This command allows you to copy a BLOB from the database into an external file. It is the responsibility of the user to ensure that the correct file type is used — don't call an image file something.txt when it should be a jpeg for example.

Blobdump requires two parameters, a blob id and a filename. The latter is simple but the former is more convoluted. You are required to pass the blob id as a pair of hexadecimal numbers, separated by a colon. The first number is the relation id number for the table in question and the second is a

sequential number within the database. You will see this pair of numbers when you select any BLOB column's data from a table — it is displayed above the BLOB contents, assuming that the display of BLOBs is turned on. See the `set blobdisplay` command below for details.

```
SQL> set blobdisplay off;

SQL> select proj_id, proj_desc
CON> from project
CON> where proj_id = 'MKTPR';

PROJ_ID         PROJ_DESC
======= =================
MKTPR               85:10

SQL> blobdump 85:10 project.jpg;

SQL> blobdump 85:10 project.txt;
```

The blob id required in the above example is the '85:10' value. You will notice that I have dumped this BLOB to both a jpeg and a text file. Isql gave no errors for the fact that I attempted to dump the BLOB to a jpeg file when the BLOB in question is text. Attempting to open the jpeg file with any image viewers will, however, result in an error. The text file opens happily in any of the assorted text viewers or editors installed on the system.

17.2. Blobview

```
SQL> BLOBVIEW blob_id;
```

This command is similar to blobdump above, but only requires the blob id parameter as the BLOB data will be displayed in an editor.

```
SQL> blobview 85:10;
```

The contents of the selected BLOB are displayed in an external editor. When the editor is closed, control returns to `isql`. You cannot use `isql` while the editor is open.

BLOBVIEW may return an "Invalid transaction handle" error after you close the editor. This is a known bug. To correct the situation, start a transaction manually, with the command `SET TRANSACTION;`

17.3. Edit

```
SQL> EDIT [filename];
```

This command allows you to edit an existing file. This may be a file of SQL commands to be used by the isql input command (see below) or any other text file. The file must, however, already exist.

If no filename is supplied, a history of all your previous commands will be displayed for editing. Please note that when you exit from the editor in this case, the commands left in the buffer at the end of the edit will be executed as a script file.

17.4. Help

The `help` command has been discussed above.

17.5. Add

```
SQL> ADD table_name;
```

This command, when passed a table name, prompts you for each column's data and adds a row to the table. You may add as many rows as you wish as the command continues until either an error occurs or the `Enter` key is pressed with no data. If you wish to set a column to `NULL`, type it in exactly as shown.

```
SQL> add country;

Enter data or NULL for each column.  RETURN to end.
Enter COUNTRY>Scotland
Enter CURRENCY>GBP

Enter COUNTRY>


SQL> commit;
```

17.6. Copy

```
SQL> COPY from_table_name to_table_name [other_database];
```

The `copy` command allows you to copy *most of* the *structure* of a table to a new table, in the current database or to a different one. Unfortunately it has a couple of problems:

- It shells out to the command line to do the work, and connects to the receiving database using an application named `isql`. If, like me, your system has renamed `isql` to `isql-fb`, you will actually end up running the wrong `isql` application and confusing error messages will be the only result.
- It assumes that `isql` will be on the `$PATH` or `%PATH%`.
- You need to define `ISC_USER` and `ISC_PASSWORD` for the child `isql` process to login to the receiving

database to create the table. This is *very* insecure.

- Because of the need for ISC_USER and ISC_PASSWORD, the receiving database must be running on the *same server* as the source database.
- The data in the table is not copied to the receiving database. Only the following parts of the table's structure is copied.
  - Domains required to recreate the table. This only applies if the copy is to another database.
  - The table itself will be created.
  - Primary key constraint, if there is one.
  - The index used to support the primary key constraint, if there is one.
- Not all of the table structure is actually copied. Missing are:
  - Foreign Key constraints.
  - Check constraints.
  - Indices other than the primary key index.
  - Triggers.
  - All of the table's data.

If you wish to copy to a different database, then the other database must be on the *same server* as the current one. You cannot, for example, connect to a database on a server named tux, and copy a table to a database running on the server tuxrep. The copy command has no way to allow you to pass a username and/or password and, equally, setting ISC_USER and ISC_PASSWORD only affects databases on the current server.

```
tux> $ export ISC_USER=SYSDBA
tux> $ export ISC_PASSWORD=secret
tux> isql employee

Database:  employee, User: sysdba

SQL> -- MAke a copy of the employee table into this database.
SQL> copy employee employee_2;


SQL> -- Compare table structures...
SQL> show table employee;

EMP_NO                          (EMPNO) SMALLINT Not Null
FIRST_NAME                      (FIRSTNAME) VARCHAR(15) Not Null
LAST_NAME                       (LASTNAME) VARCHAR(20) Not Null
PHONE_EXT                       VARCHAR(4) Nullable
HIRE_DATE                       TIMESTAMP Not Null DEFAULT 'NOW'
DEPT_NO                         (DEPTNO) CHAR(3) Not Null
                                CHECK (VALUE = '000' OR
                                (VALUE > '0' AND VALUE <= '999') OR VALUE IS NULL)
JOB_CODE                        (JOBCODE) VARCHAR(5) Not Null
```

```
                                CHECK (VALUE > '99999')
JOB_GRADE                       (JOBGRADE) SMALLINT Not Null
                                CHECK (VALUE BETWEEN 0 AND 6)
JOB_COUNTRY                     (COUNTRYNAME) VARCHAR(15) Not Null
SALARY                          (SALARY) NUMERIC(10, 2) Not Null DEFAULT 0
                                CHECK (VALUE > 0)
FULL_NAME                       Computed by: (last_name || ', ' || first_name)

CONSTRAINT INTEG_28:
  Foreign key (DEPT_NO)    References DEPARTMENT (DEPT_NO)
CONSTRAINT INTEG_29:
  Foreign key (JOB_CODE, JOB_GRADE, JOB_COUNTRY)
  References JOB (JOB_CODE, JOB_GRADE, JOB_COUNTRY)
CONSTRAINT INTEG_27:
  Primary key (EMP_NO)
CONSTRAINT INTEG_30:
  CHECK ( salary >= (SELECT min_salary FROM job WHERE
                        job.job_code = employee.job_code AND
                        job.job_grade = employee.job_grade AND
                        job.job_country = employee.job_country) AND
            salary <= (SELECT max_salary FROM job WHERE
                        job.job_code = employee.job_code AND
                        job.job_grade = employee.job_grade AND
                        job.job_country = employee.job_country))

Triggers on Table EMPLOYEE:
SET_EMP_NO, Sequence: 0, Type: BEFORE INSERT, Active
SAVE_SALARY_CHANGE, Sequence: 0, Type: AFTER UPDATE, Active


SQL> show table employee_2;

EMP_NO                          (EMPNO) SMALLINT Not Null
FIRST_NAME                      (FIRSTNAME) VARCHAR(15) Not Null
LAST_NAME                       (LASTNAME) VARCHAR(20) Not Null
PHONE_EXT                       VARCHAR(4) Nullable
HIRE_DATE                       TIMESTAMP Not Null DEFAULT 'NOW'
DEPT_NO                         (DEPTNO) CHAR(3) Not Null
                                CHECK (VALUE = '000' OR
                                (VALUE > '0' AND VALUE <= '999') OR VALUE IS NULL)
JOB_CODE                        (JOBCODE) VARCHAR(5) Not Null
                                CHECK (VALUE > '99999')
JOB_GRADE                       (JOBGRADE) SMALLINT Not Null
                                CHECK (VALUE BETWEEN 0 AND 6)
JOB_COUNTRY                     (COUNTRYNAME) VARCHAR(15) Not Null
SALARY                          (SALARY) NUMERIC(10, 2) Not Null DEFAULT 0
                                CHECK (VALUE > 0)
FULL_NAME                       Computed by: (last_name || ', ' || first_name)

CONSTRAINT INTEG_93:
  Primary key (EMP_NO)
```

```
SQL> -- Check indices on both tables...
SQL> show indices employee;

NAMEX INDEX ON EMPLOYEE(LAST_NAME, FIRST_NAME)
RDB$FOREIGN8 INDEX ON EMPLOYEE(DEPT_NO)
RDB$FOREIGN9 INDEX ON EMPLOYEE(JOB_CODE, JOB_GRADE, JOB_COUNTRY)
RDB$PRIMARY7 UNIQUE INDEX ON EMPLOYEE(EMP_NO)


SQL> show indices employee_2;
RDB$PRIMARY27 UNIQUE INDEX ON EMPLOYEE_2(EMP_NO)


SQL> -- Check data counts on both tables...
SQL> select count(*) from employee;

       COUNT
============
          42


SQL> select count(*) from employee_2;

       COUNT
============
           0
```

The `copy` command only works provided your `isql` application is really named `isql`. In addition, if you have lots of data in the table, you still have to copy it manually as the `copy` command will only copy the table structure. Remember that the new table will have no triggers, no foreign keys, no indices — other than the primary key one — and no data.

It is possible that the `copy` command will be removed from `isql` at some future release.

17.7. Input

```
SQL> INput filename;
```

This command enables the user to execute a number of commands from a script file rather than manually typing them all into `isql` at the prompt. The script may contain any mix of DDL and/or DDL commands, along with `isql` commands to redirect output, change options, etc.

```
SQL> shell;

$ cat test.sql
drop table fred;
commit;

$ exit;


SQL> show table fred;

A                               INTEGER Nullable
B                               INTEGER Not Null


SQL> input test.sql;


SQL> show table fred;
There is no table FRED in this database
```

17.8. Output

```
SQL> OUTput [filename];
```

This command redirects all output that normally is displayed on the screen, to a specific file. If a file name is supplied, all subsequent output goes to that file and is not displayed on screen. If no file name is supplied, output is once more redirected to the screen.

```
SQL> output test.log;

SQL> show tables;

SQL> output;

SQL> shell;

$ cat test.log

       COUNTRY                         CUSTOMER
       DEPARTMENT                      EMPLOYEE
       EMPLOYEE_PROJECT                FRED
       JOB                             PROJECT
       PROJ_DEPT_BUDGET                SALARY_HISTORY
       SALES
```

17.9. Set

There are a number of settings and options that can be changed to suit how you wish isql to operate. These settings are changed by the set command which is discussed below.

17.10. Shell

```
SQL> SHELL;
```

This command allows you to temporarily exit from isql and use a shell session to carry out some further processing. On exiting from the shell, you will return to isql. You cannot use the isql session that activated the shell while the shell session remains open.

```
SQL> shell;

$ cat test.log

       COUNTRY                        CUSTOMER
       DEPARTMENT                     EMPLOYEE
       EMPLOYEE_PROJECT               FRED
       JOB                            PROJECT
       PROJ_DEPT_BUDGET               SALARY_HISTORY
       SALES

$ exit

SQL>
```

17.11. Show

There are a number of settings and options that can be changed to suit how you wish isql to operate. The show command allows you to view the way that these have been set up by the set commands, or by other options. These are discussed below.

17.12. Exit

```
SQL> EXIT;
```

The exit command will commit any uncommitted work and exit from isql.

17.13. Quit

```
SQL> QUIT;
```

The quit command will rollback any uncommitted work and exit from `isql`.

Chapter 18. Isql Set Commands

As explained in the help command, you may enter the help set command to drill down into the various options available for the set command. These are all discussed below. Note that the output from the help set command does not include the set transaction command. The help set command produces the following output (from Firebird 2.5):

```
SQL> help set;

Set commands:
    SET                     -- display current SET options
    SET AUTOddl             -- toggle autocommit of DDL statements
    SET BAIL                -- toggle bailing out on errors in non-interactive mode
    SET BLOB [ALL|<n>]      -- display BLOBS of subtype <n> or ALL
    SET BLOB                -- turn off BLOB display
    SET COUNT               -- toggle count of selected rows on/off
    SET ROWCOUNT [<n>]      -- limit select stmt to <n> rows, zero is no limit
    SET ECHO                -- toggle command echo on/off
    SET HEADING             -- toggle display of query column titles
    SET LIST                -- toggle column or table display format
    SET NAMES <csname>      -- set name of runtime character set
    SET PLAN                -- toggle display of query access plan
    SET PLANONLY            -- toggle display of query plan without executing
    SET SQL DIALECT <n>     -- set sql dialect to <n>
    SET STATs               -- toggle display of performance statistics
    SET TIME                -- toggle display of timestamp with DATE values
    SET TERM <string>       -- change statement terminator string
    SET WIDTH <col> [<n>]   -- set/unset print width to <n> for column <col>

All commands may be abbreviated to letters in CAPitals
```

In the above, the BLOB commands are incomplete. They should be BLOBdisplay. The above is displayed when the set command is executed with no parameters, however, in the following descriptions of the various set commands, I will be using the full BLOBdisplay version of the appropriate commands.

The last line of the above output indicates that these commands can be abbreviated to the letters in capitals. Unfortunately, other than the set autoddl command, none of the others appear to have a short form.

18.1. Set

The set command, with no parameters, displays the current settings, as the following example from Firebird 2.5 shows:

```
SQL> set;

Set commands:
    SET                    -- display current SET options
    SET AUTOddl            -- toggle autocommit of DDL statements
    SET BAIL               -- toggle bailing out on errors in non-interactive mode
    SET BLOB [ALL|<n>]     -- display BLOBS of subtype <n> or ALL
    SET BLOB               -- turn off BLOB display
    SET COUNT              -- toggle count of selected rows on/off
    SET ROWCOUNT [<n>]     -- limit select stmt to <n> rows, zero is no limit
    SET ECHO               -- toggle command echo on/off
    SET HEADING            -- toggle display of query column titles
    SET LIST               -- toggle column or table display format
    SET NAMES <csname>     -- set name of runtime character set
    SET PLAN               -- toggle display of query access plan
    SET PLANONLY           -- toggle display of query plan without executing
    SET SQL DIALECT <n>    -- set sql dialect to <n>
    SET STATs              -- toggle display of performance statistics
    SET TIME               -- toggle display of timestamp with DATE values
    SET TERM <string>      -- change statement terminator string
    SET WIDTH <col> [<n>]  -- set/unset print width to <n> for column <col>
```

18.2. Set Autoddl

```
SQL> SET AUTOddl [on | off];
```

This command sets whether all DDL statements executed will be automatically committed or not. The command without any parameters acts as a toggle and turns autoddl off if it is currently on and vice versa. You may supply a specific parameter to make your intentions clear. The parameter must be one of `on` or `off`. The `set` command, with no parameters, will display the current setting. The default in `isql` is equivalent to `set autoddl on`.

18.3. Set Bail

```
SQL> SET BAIL [on | off];
```

Setting this command determines whether or not `isql` will "bail out" on any errors when the input command has been used to read a script file. `Isql` will not exit if it is running in interactive mode, and you cause an error.

Executing this command, without passing a parameter, results in a toggling of the current state. If `bail` is on, it will turn off and vice versa.

18.4. Set Blobdisplay

```
SQL> SET BLOBdisplay [n | all | off];
```

This command determines if BLOB column data is to be displayed in the output when a table with BLOB columns is queried. The default for this command, if no parameters are passed, is to set BLOB data off — it will not be displayed, only the blob id will be shown.

The blob id is discussed above in the section describing the blobdump and blobview commands.

If all is passed, then all BLOB sub-types will be displayed.

If a number representing the blob sub-type is passed, then only BLOBs with the specififc sub-type will be displayed. The default is 1 for text sub-types.

```
SQL> -- Don't display any blob data.
SQL> set blob off;

SQL> select proj_desc
CON> from project
CON> where proj_id = 'HWRII';

        PROJ_DESC
=================
             85:e


SQL> -- Display all blob data.
SQL> set blobdisplay all;

SQL> select proj_desc
CON> from project
CON> where proj_id = 'HWRII';

        PROJ_DESC
=================
             85:e
================================================================================
PROJ_DESC:
Integrate the hand-writing recognition module into the
universal language translator.
================================================================================


SQL> -- Only display type 1 blob data = text.
SQL> set blob 1;

SQL> select proj_desc
CON> from project
```

```
CON> where proj_id = 'HWRII';

        PROJ_DESC
=================
             85:e
================================================================================
PROJ_DESC:
Integrate the hand-writing recognition module into the
universal language translator.
================================================================================


SQL> -- Only display blob type 7 = not text!
SQL> set blob 7;

SQL> select proj_desc
CON> from project
CON> where proj_id = 'HWRII';

        PROJ_DESC
=================
             85:e
================================================================================
PROJ_DESC:
BLOB display set to subtype 7. This BLOB: subtype = 1
================================================================================
```

You will notice in the last example that a message was displayed advising that we are only displaying BLOB data for sub-type 7 and the BLOB data in this table is a sub-type 1, so the data are not displayed.

18.5. Set Count

```
SQL> SET COUNT [on | off];
```

This command determines whether a line of text is displayed at the end of the output from a DML statement, telling the user how many rows were affected.

```
SQL> set count on;

SQL> select count(*) from employee;

       COUNT
============
          42

Records affected: 1
```

The record count is displayed for all DDL operations, not just for a SELECT.

```
SQL> create table fred( a integer);
SQL> commit;

SQL> insert into fred values (666);
Records affected: 1

SQL> update fred set a = 123 where a = 666;
Records affected: 1

SQL> delete from fred;
Records affected: 1

SQL> commit;
```

18.6. Set Rowcount

```
SQL> SET ROWCOUNT [n];
```

Setting rowcount to zero, which is the default when isql is started, results in a select statement returning all rows which meet the criteria in the where clause. There are circumstances where you do not want lots and lots of output scrolling up the screen, so you may set rowcount to a smaller number and all subsequent select statements will only display the first *n* rows instead of everything.

```
SQL> set count on;
SQL> set rowcount 0;

SQL> select emp_no from employee;

 EMP_NO
=======
      2
      4
...
    144
    145

Records affected: 42


SQL> set rowcount 10;
SQL> select emp_no from employee;

 EMP_NO
=======
      2
      4
...
     15
     20

Records affected: 10
```

There is no indication that `rowcount` is restricting the number of rows returned, it is the responsibility of the user to remember, or check whether `rowcount` is on or off. Using `rowcount` can lead to confusion about exactly how many rows there are in a table!

18.7. Set Echo

```
SQL> SET ECHO [ON | OFF];
```

The default is on if you do not supply a value. This command causes all the SQL commands being executed to be displayed on the output device prior to their execution. You may wish to turn echo off as part of a script file although the `isql` default is for echo to be off.

```
SQL> set echo on;

SQL> select count(*) from rdb$database;
select count(*) from rdb$database;

       COUNT
============
           1


SQL> set echo off;
set echo off;

SQL> select count(*) from rdb$database;

       COUNT
============
           1
```

This command can be handy in a script file. If you receive an error, it can sometimes be difficult to determine the exact SQL statement that caused it. If you `set echo on` in your script, you will at least be able to determine exactly which statement failed.

18.8. Set Heading

```
SQL> SET HEADING [ON | OFF];
```

This command turns the display of column headings on or off as desired. If no parameter is supplied to the command, it toggles the current state of the heading display.

```
SQL> set heading off;

SQL> select count(*) from employee;

          42


SQL> set heading on;

SQL> select count(*) from employee;

       COUNT
============
          42
```

18.9. Set List

```
SQL> SET LIST [ON | OFF];
```

This command controls how the data returned by a select statement will be displayed. The default setting is to display the data in tabular form with optional column headings at the top of each 'page'. Setting the list mode to on results in a different format where each column heading is displayed on the left and the column data on the right. This repeats for each and every row returned by the query.

As with other commands, not providing a value to the command results in a toggle of the current setting.

```
SQL> set list off;

SQL> select emp_no, first_name, last_name, salary
CON> from employee;

 EMP_NO FIRST_NAME      LAST_NAME                      SALARY
======= =============== ==================== ====================
      2 Robert          Nelson                       105900.00
      4 Bruce           Young                         97500.00
      5 Kim             Lambert                      102750.00
      8 Leslie          Johnson                       64635.00
...


SQL> set list on;

SQL> select emp_no, first_name, last_name, salary
CON> from employee;

EMP_NO                          2
FIRST_NAME                      Robert
LAST_NAME                       Nelson
SALARY                          105900.00

EMP_NO                          4
FIRST_NAME                      Bruce
LAST_NAME                       Young
SALARY                          97500.00
...
```

18.10. Set Names

```
SQL> SET NAMES [character_set];
```

This command defines the character set to be used in subsequent database transactions. If the default database charcter set is not NONE, then in situations where the client uses a different character set to the database, it is possible to suffer from data corruption as some character sets cannot convert some characters to a suitable character in another character set.

If you don't pass a character set, the default will be to use the NONE character set.

You can determine a list of the valid character sets to use with the following query:

```
SQL> set width RDB$CHARACTER_SET_NAME 30;

SQL> select RDB$CHARACTER_SET_NAME
CON> from RDB$CHARACTER_SETS
CON> order by 1;

RDB$CHARACTER_SET_NAME
==============================
ASCII
BIG_5
CP943C
CYRL
DOS437
...
ISO8859_1
ISO8859_13
...
NONE
OCTETS
...
UTF8
...
WIN1258
```

18.11. Set Plan

```
SQL> SET PLAN [ON | OFF];
```

This command determines whether or not isql will display the plan it used to access the data for each statement executed. The isql default is never to display the plan. As with many other commands, not providing a parameter toggles the current state.

```
SQL> set plan on;

SQL> select emp_no, first_name, last_name
CON> from employee
CON> where emp_no = 107;

PLAN (EMPLOYEE INDEX (RDB$PRIMARY7))

 EMP_NO FIRST_NAME      LAST_NAME
======= =============== ====================
    107 Kevin           Cook


SQL> update employee
CON> set first_name = 'Norman'
CON> where last_name = 'Cook';

PLAN (EMPLOYEE INDEX (NAMEX))


SQL> select count(*) from employee;

PLAN (EMPLOYEE NATURAL)

       COUNT
============
          42
```

The execution plan is displayed before the output from a select statement.

18.12. Set Planonly

```
SQL> SET PLANONLY [ON | OFF];
```

This command prevents Firebird from actually executing the SQL statement and instead, simply shows the plan that it would use to access the data. This command relies on the `set plan` command. If `set plan off` had been executed, this command would have no effect, so turning `planonly` on has the additional effect of executing `set plan on` implicitly. Executing `set planonly off` does *not* implicitly execute `set plan off`.

```
SQL> set planonly on;

SQL> select count(*) from employee;

PLAN (EMPLOYEE NATURAL)
```

As before, not supplying a parameter toggles the current setting.

18.13. Set SQLDA_Display

This is a hidden command which is not mentioned in the output from the `help set` command. It displays internal details about the SQL statements being executed by `isql`. This used to be only available in a special debug build, but since version 2.0, it is available in `isql`.

```
SQL> set sqlda_display on;

SQL> select count(*) from employee;

INPUT  SQLDA version: 1 sqln: 10 sqld: 0

OUTPUT SQLDA version: 1 sqln: 20 sqld: 1
01: sqltype: 496 LONG                    sqlscale: 0 sqlsubtype: 0 sqllen: 4
  :  name: (5)COUNT  alias: (5)COUNT
  : table: (0)  owner: (0)

       COUNT
============
          42
```

Note that when you run the `help set` or `set` commands, no information about this command will be displayed.

18.14. Set SQL Dialect

```
SQL> SET SQL DIALECT {1 | 2 | 3};
```

This command specifies the Firebird SQL dialect to which the client session is to be changed. If the session is currently attached to a database of a different dialect to the one specified in the command, a warning is displayed. The values permitted are:

- 1 — which sets the client connection to SQL dialect 1
- 2 — which sets the client connection to SQL dialect 2.
- 3 — which sets the client connection to SQL dialect 3.

See Dialects for details of the differences between the three dialects.

```
SQL> set sql dialect 1;
WARNING: Client SQL dialect has been set to 1 when
connecting to Database SQL dialect 3 database.
...
SQL> set sql dialect 3;
SQL>
```

The warning in the above example has had to be split over two lines in order to have it fit on the page. Normally, it consist of a single line.

18.15. Set Stats

```
SQL> SET STATs [ON | OFF];
```

This command determines whether or not isql should display various statistics about each SQL command executed. As usual, failing to pass a parameter results in the current setting being toggled.

```
SQL> set stats on;

SQL> select count(*) from employee;

       COUNT
============
          42

Current memory = 10094216
Delta memory = 16
Max memory = 10227608
Elapsed time= 0.00 sec
Cpu = 0.00 sec
Buffers = 2048
Reads = 0
Writes = 0
Fetches = 92
```

18.16. Set Time

```
SQL> SET TIME [ON | OFF];
```

This command applies to dialect 1 databases only. It causes the time portion to be displayed or not, when the selected data is a column defined with the DATE data type. It has no effect in other dialects.

18.17. Set Term

```
SQL> SET TERM new_terminator current_terminator
```

This command changes the default statement terminator from a semi-colon to something else as defined in the passed string. This is mostly useful when you are about to enter a string of SQL statements making up a procedure, for example, or a trigger. Isql would attempt to execute each statement when it sees a terminating semi-colon, so you would change the terminator first, then enter the required code. When complete, you would change it back, but when doing so, you must remember to terminate the `set term` command with the *current* terminating character(s).

When first started, `isql` uses the semi-colon as the default terminator.

You can, if desired, simply change the terminator because you prefer something other than a semi-colon. You don't have to be writing procedures in order to change it.

```
SQL> -- Change terminator from ; to +
SQL> set term +;

SQL> select count(*) from employee+

       COUNT
============
          42

SQL> -- Change terminator from + to 'fred'
SQL> set term fred +

SQL> select count(*) from employee fred

       COUNT
============
          42

SQL> -- Change back from 'fred' to ;
SQL> set term ; fred
```

See the section on the terminator for full details.

18.18. Set Transaction

This is another hidden command which is not mentioned in the output from the `help set` command.

There is a default transaction started for you when you use isql. When you commit or rollback in isql, the default transaction ends, and a new default transaction begins. These transactions are:

- READ WRITE — meaning that any SQL statment that is executed may make changes in the database.
- WAIT — meaning that if a row in a table is currently locked by another session, the execution of the statement will appear to hang until the other session either commits or rolls back.
- SNAPSHOT — meaning that this transaction will be guaranteed a non-volatile view of the data and will be unaffected by any changes made and committed in any other transactions that take place while this one remains unfinished by a commit or rollback.

A full explanation of transactions is beyond the scope of this manual. For more information see the *Firebird 2.5 Language Reference*, or *The Firebird Book* by Helen Borrie.

18.19. Set Warnings

```
SQL> SET {WARNINGS | WNG} [ON | OFF];
```

This command specifies whether warnings are to be output. A few examples for which isql issues warnings are:

- SQL statements with no effect.
- Pending database shutdown.
- API calls that may be replaced in future versions of Firebird.
- Expressions that may cause differing results in different versions of Firebird.
- In Firebird 1.0, SQL statements with ambiguous join specifications. More recent Firebird versions will raise an exception rather than a warning.

As with many of the set commands, set warnings acts as a toggle if no parameter is supplied.

18.20. Set Width

Normally the width of a *character* column in a table defines the width of the output when that column is selected. Using the set width command allows the user to define a wider or narrower output column width.

The format of the command is:

```
SQL> SET WIDTH column_or_alias [width];
```

The setting remains until changed to a new width, or until cancelled by the set width column_or_alias; command — no width supplied means use the default width setting for this column.

The following example shows the width of the last_name column being amended. The first SELECT shows the default setting which is a wdith of 20 characters (count the '=' in the headings) which is the definition of the last_name column in the employee table. The second shows the width being

reduced to 10 characters.

```
SQL> select first 10 emp_no, last_name
CON> from employee
CON> order by last_name;

 EMP_NO LAST_NAME
======= ====================
     34 Baldwin
    105 Bender
     28 Bennet
     83 Bishop
    109 Brown


SQL> set width last_name 10;

SQL> select first 10 emp_no, last_name
CON> from employee
CON> order by last_name;

 EMP_NO LAST_NAME
======= ==========
     34 Baldwin
    105 Bender
     28 Bennet
     83 Bishop
    109 Brown
```

EMP_NO is a smallint data type. Unfortunately, it doesn't appear to be possible to change the width on non-character columns like integer, smallint etc. The `set width emp_no 10;` command, for example, has no effect, as shown below, which also demonstrates turning off a previous width setting for the last_name column:

```
SQL> set width last_name;

SQL> set width emp_no 10;

SQL> select first 10 emp_no, last_name
CON> from employee
CON> order by last_name;

 EMP_NO LAST_NAME
======= ====================
     34 Baldwin
    105 Bender
     28 Bennet
     83 Bishop
    109 Brown
```

Chapter 19. Isql Show commands

As explained in the help command, there are a number of individual show commands within isql. The general format of the show commands is:

```
SQL> SHOW [<object> [name]] ;
```

The *object* is always required and the *name* is required to display details of a specific object. Without a name, the commands will normally display all the objects of the requested type.

Unfortunately, unlike the set commands, there is no handy drill down into the various show commands using the help command. However, if you type show on its own, you will be given a little more assistance.

```
SQL> show;

Valid options are:
CHECKs                  COMMENTs                COLLATEs
COLLATIONs              DOMAINs                 DB
DATABASE                DEPENdency              DEPENdencies
EXCEPtions              FILTERs                 FUNCtions
GENerators              GRANTs                  INDexes
INDICES                 PROCedures              ROLEs
SYStem                  SEQuences               SECURITY CLAsses
SECCLAsses              TABLEs                  TRIGgers
USERS                   VIEWs
Command error: show
```

The upper case letters indicate what you must type as an absolute minimum.

The show commands are detailed and described below. Where possible, examples from the employee database are shown.

19.1. Show Checks

```
SQL> SHOW CHECKs table_name;
```

This command displays all user-defined check constraints defined for a specific table. Unlike other show commands, there is no option to display a list of all the check constraints in the database. You must always provide a table name as part of the command.

```
SQL> show check employee;

CONSTRAINT INTEG_30:
  CHECK ( salary >= (SELECT min_salary FROM job WHERE
                         job.job_code = employee.job_code AND
                         job.job_grade = employee.job_grade AND
                         job.job_country = employee.job_country) AND
             salary <= (SELECT max_salary FROM job WHERE
                         job.job_code = employee.job_code AND
                         job.job_grade = employee.job_grade AND
                         job.job_country = employee.job_country))
```

19.2. Show Collations

```
SQL> SHOW {COLLATIONs | COLLATEs} [name];
```

These commands display a list of all the user defined collations in the current database. It is only available from Firebird 2.0 onwards. The first form of the commands display a list of all the collations while a specific collation may be displayed by providing the collation name.

```
SQL> show collations;
UNICODE_ENUS_CI, CHARACTER SET UTF8, FROM EXTERNAL ('UNICODE'), PAD SPACE,
CASE INSENSITIVE, 'COLL-VERSION=58.0.6.48'
UNICODE_ENUS_CS, CHARACTER SET UTF8, FROM EXTERNAL ('UNICODE'), PAD SPACE,
'COLL-VERSION=58.0.6.48'

SQL> show collation unicode_enus_ci;
UNICODE_ENUS_CI, CHARACTER SET UTF8, FROM EXTERNAL ('UNICODE'), PAD SPACE,
CASE INSENSITIVE, 'COLL-VERSION=58.0.6.48'
```

You can see from the output above, which is not part of the employee database, does appear to display all the relevant information in the first form of the command. There does not appear to be much reason to drill down into a specific collation — at least, not according to this example. Some lines in the above have had to be split over two to allow it to fit on the page.

Show Comments

```
SQL> SHOW COMMENTs;
```

This command displays all comments that have been created, on various objects, in the current database. There is no option to display a specific comment. Each comments is listed along with the object type and name, to whihc it has been applied.

```
SQL> show comments;

COMMENT ON DATABASE IS This is the demonstration EMPLOYEE database.;
COMMENT ON TABLE EMPLOYEE IS The EMPLOYEE table has details of our employees.;
```

The actual comment text is shown between the word 'IS' and the trailing semicolon.

19.3. Show Database

```
SQL> SHOW {DATABASE | DB};
```

The `show database` (or `show db`) command displays details about the *current* database. The ODS version, shown in the following examples, is only displayed from Firebird version 2.0 onwards.

```
SQL> show database;

Database: employee
        Owner: SYSDBA
PAGE_SIZE 4096
Number of DB pages allocated = 270
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 190
Transaction - oldest active = 191
Transaction - oldest snapshot = 191
Transaction - Next = 211
ODS = 11.2
Default Character set: NONE
```

No parameters, such as a specific database name, are required and if supplied, will be ignored. The details displayed will always be for the current database.

```
SQL> show database testing_db;

Database: employee
        Owner: SYSDBA
PAGE_SIZE 4096
...
Default Character set: NONE
```

You will note from the above that the details displayed are still for the employee database.

19.4. Show Dependencies

```
SQL> SHOW {DEPENdencies | DEPENdency} object_name;
```

These commands display all dependencies for the specified object name supplied as a parameter. The object name supplied need not necessarily be a table name, it could be a function or procedure name, a sequence name etc.

The output listed is a comma separated list of the other objects in the database *upon which* the supplied object is dependent. In other words, a procedure would fail to compile if any of the listed dependencies was to be removed, for example.

```
SQL> show dependencies SET_CUST_NO;

        [SET_CUST_NO:Trigger]
CUSTOMER:Table<-CUST_NO, CUST_NO_GEN:Generator
+++
```

The listing above shows that `SET_CUST_NO` is a trigger and that it is dependent on two separate objects, the `CUST_NO` column of table `CUSTOMER` and the sequence/generator named `CUST_NO_GEN`. If you display the trigger itself, you will see both of those objects mentioned:

```
SQL> show trigger set_cust_no;

Triggers on Table CUSTOMER:
SET_CUST_NO, Sequence: 0, Type: BEFORE INSERT, Active
AS
BEGIN
    if (new.cust_no is null) then
    new.cust_no = gen_id(cust_no_gen, 1);
END
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Sometimes, the output can be a little confusing. You may see various objects in the list that don't appear to be relevant. The `RDB$DEPENDENCIES` table, where the data comes from, also holds details of system objects upon which a given object will depend.

19.5. Show Domains

```
SQL> SHOW DOMAINs [name];
```

This command displays domain information. A domain is a user-defined data type, global to the database. It is used to define the format and range of columns, upon which the actual column definitions in tables are based.

Firebird tables are defined by the specification of columns, which store appropriate information in each column using data types.

A data type is an elemental unit when defining data, which specifies the type of data stored in tables, and which operations may be performed on this data. It can also include permissible calculative operations and maximum data size. Examples of data types include: numerical (numeric, decimal, integer);textual (char, varchar, nchar, nvarchar); date (date, time, timestamp) and blobs(binary large objects).

As with many show commands, there are two forms. The first displays a list of all known domains in the database while the second allows you to display the details of a specific domain.

```
SQL> show domain;

       ADDRESSLINE                             BUDGET
       COUNTRYNAME                             CUSTNO
       DEPTNO                                  EMPNO
       FIRSTNAME                               JOBCODE
...

SQL> show domain addressline;
ADDRESSLINE                     VARCHAR(30) Nullable
```

19.6. Show Exceptions

```
SQL> SHOW EXCEPtions [name];
```

This command displays all the exceptions which have been defined in the current database. Details of the exception's error message and objects which use the exception — those which are dependant upon the exception — are also shown. You may display individual exception's details with the second for of the command.

```
SQL> show exceptions;

Exception Name                  Used by, Type
=============================== ========================================
CUSTOMER_CHECK                  SHIP_ORDER, Stored procedure
Msg: Overdue balance -- can not ship.

CUSTOMER_ON_HOLD                SHIP_ORDER, Stored procedure
Msg: This customer is on hold.

...


SQL show exception customer_on_hold;

Exception Name                  Used by, Type
=============================== ========================================
CUSTOMER_ON_HOLD                SHIP_ORDER, Stored procedure
Msg: This customer is on hold.
```

19.7. Show Filters

```
SQL> SHOW FILTERs [name];
```

This command displays a list of all known BLOB filters declared in the current database using the declare filter command. The second form of the command allows the full details of a specific filter to be displayed.

```
SQL> show filter;

       FUNNEL
       ...


SQL> show filter funnel;
BLOB Filter: FUNNEL
        Input subtype: 2 Output subtype: 1
        Filter library is myfilterlib
        Entry point is blr2asc
```

19.8. Show Functions

```
SQL> SHOW FUNCtions [name];`
```

This command allows a list of all external functions declared in the current database, to be displayed. External functions are those defined and coded in various UDF libraries.

The second form of the command allows the details of a specific function to be displayed.

```
SQL> show functions;

       ADDDAY                          ADDDAY2
       ADDHOUR                         ADDMILLISECOND
       ADDMINUTE                       ADDMONTH
       ADDSECOND                       ADDWEEK
       ADDYEAR


SQL> show function addyear;

Function ADDYEAR:
Function library is fbudf
Entry point is addYear
Returns  TIMESTAMP
Argument 1: TIMESTAMP
Argument 2: INTEGER
```

19.9. Show Generators

```
SQL> SHOW {GENERATORs | SEQuences} [name];
```

`SHOW GENERATORs` and `SHOW SEQuences` are identical. Generators was the old Firebird term for what are more commonly known as sequences in other databases, as well as the ANSI Standards. You are encouraged to use sequences rather than generators but `isql` considers them to be the same.

The first form of the commands above list all the sequences in the current database, while the second form displays details of a specific sequence.

```
SQL> show sequences;

Generator CUST_NO_GEN, current value is 1015
Generator EMP_NO_GEN, current value is 145


SQL> show sequence emp_no_gen;

Generator EMP_NO_GEN, current value is 145
```

19.10. Show Grants

```
SQL> SHOW GRANTs [{object_name | role_name}];
```

This command displays a list of all grants in the current database if the first format of the command is used. The second drills down and displays only those details for the selected object, which may be a table, procedure, etc. Alternatively, if a role name is provided, only a list of users who have been granted that role will be displayed.

```
SQL> show grants;

/* Grant permissions for this database */
GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES
ON COUNTRY TO PUBLIC WITH GRANT OPTION
GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES
ON CUSTOMER TO PUBLIC WITH GRANT OPTION
...
GRANT SELECT ON EMPLOYEE TO ROLE DEFAULT_USER
...
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ TO PUBLIC WITH GRANT OPTION
GRANT EXECUTE ON PROCEDURE ALL_LANGS TO PUBLIC WITH GRANT OPTION
...


SQL> show grants employee;

GRANT DELETE, INSERT, SELECT, UPDATE, REFERENCES
ON EMPLOYEE TO PUBLIC WITH GRANT OPTION


SQL> show grants ship_order;

GRANT EXECUTE ON PROCEDURE SHIP_ORDER TO PUBLIC WITH GRANT OPTION


SQL> show grants default_user;

GRANT DEFAULT_USER TO SYSDBA
```

Note that some lines in the above have been split to allow them to fit on the page.

19.11. Show Indexes

```
SQL> SHOW {INDexes | INDICES} [{table_name | index_name}]
```

You may specify either `indexes` (or abbreviated forms starting with `ind`) or `indices`, they are treated

as identical by Firebird. The first form of this command will list all the indexes in the current database. The second form of the command will display the list of indices for a specific table as determined by the table_name parameter. The final form of the command displays details of a given index and in this form of the command.

```
SQL> show indices;

RDB$PRIMARY1 UNIQUE INDEX ON COUNTRY(COUNTRY)
CUSTNAMEX INDEX ON CUSTOMER(CUSTOMER)
...
SALESTATX INDEX ON SALES(ORDER_STATUS, PAID)


SQL> show indices employee;

NAMEX INDEX ON EMPLOYEE(LAST_NAME, FIRST_NAME)
RDB$FOREIGN8 INDEX ON EMPLOYEE(DEPT_NO)
RDB$FOREIGN9 INDEX ON EMPLOYEE(JOB_CODE, JOB_GRADE, JOB_COUNTRY)
RDB$PRIMARY7 UNIQUE INDEX ON EMPLOYEE(EMP_NO)


SQL> show index namex;

NAMEX INDEX ON EMPLOYEE(LAST_NAME, FIRST_NAME)
```

19.12. Show Procedures

```
SQL> SHOW PROCedures [name];
```

This command allows a list of all procedures created in the current database, to be displayed. The second form of the command allows the details and source code to be shown for a specific procedure. See also the `show functions` and `show triggers` commands.

```
SQL> show procedures;

Procedure Name                  Invalid Dependency, Type
=============================== ======= ===================================
ADD_EMP_PROJ                            EMPLOYEE_PROJECT, Table
                                        UNKNOWN_EMP_ID, Exception
ALL_LANGS                               JOB, Table
                                        SHOW_LANGS, Procedure
...


SQL> show procedure all_langs;
Procedure text:
==============================================================================
    BEGIN
        FOR SELECT job_code, job_grade, job_country FROM job
                INTO :code, :grade, :country

        DO
        BEGIN
            FOR SELECT languages FROM show_langs
                    (:code, :grade, :country) INTO :lang DO
                SUSPEND;
            /* Put nice separators between rows */
            code = '=====';
            grade = '=====';
            country = '===============';
            lang = '==============';
            SUSPEND;
        END
    END
==============================================================================
Parameters:
CODE                            OUTPUT VARCHAR(5)
GRADE                           OUTPUT VARCHAR(5)
COUNTRY                         OUTPUT VARCHAR(15)
LANG                            OUTPUT VARCHAR(15)
```

19.13. Show Roles

```
SQL> SHOW ROLEs [name];`
```

This command lists all the roles in the current database if the first form is used or, drills down to display a list of all the users who have been granted a specific role if the second form of the command is used.

```
SQL> show roles;

       DEFAULT_USER


SQL> show role default_user;

Role DEFAULT_USER is granted to:

SYSDBA
```

19.14. Show Secclasses

```
SQL> SHOW SECCLAsses object_name;
```

This command displays details about the security classes for a given object. The object_name passed to the command need not be a table name, the command works for tables, procedures etc.

```
SQL> show secclasses employee;

Table's main sec class SQL$7
Table's default sec class SQL$DEFAULT7
```

19.15. Show Security Classes

```
SQL> SHOW SECURITY CLAsses name;
```

This command always returns an error.

```
SQL> show security classes;
Command error: show security classes

SQL> show security classes employee;
Command error: show security_classes employee
```

19.16. Show Sequences

The Firebird specific name, *generator*, has been updated to match the ANSI standard term *sequence*. The `show sequences` command is identical to `show generators` (above) and the output is identical.

19.17. Show SQL Dialect

```
SQL> SHOW SQL DIALECT;
```

This command, which must be entered in full, shows the current database's dialect as well as the dialect used by the currently connected client.

```
SQL> show SQL Dialect;
        Client SQL dialect is set to: 3 and database SQL dialect is: 3
```

19.18. Show System

```
SQL> SHOW SYStem [TABLES];
```

This command lists the internal, ie system, objects created and used in the current database. The optional parameter — `TABLES` — restricts the listing to show only tables. This applies from Firebird 2.0 onwards. Prior to version 2.0, the command would only list the system tables — equivalent to the `show system tables` command.

If no parameter is passed, the listing will display tables, functions (internal as opposed to external ones) and collations.

```
SQL> show system;
Tables:
       MON$ATTACHMENTS                         MON$CALL_STACK
...
       RDB$USER_PRIVILEGES                     RDB$VIEW_RELATIONS

Functions:
       RDB$GET_CONTEXT                         RDB$SET_CONTEXT

Collations:
       ASCII                                   BIG_5
...
       WIN1258                                 WIN_CZ
       WIN_CZ_CI_AI                            WIN_PTBR
```

If you wish to drill down and display details of a specific object, simply use the corresponding `show` command.

```
SQL> show table mon$io_stats;

MON$STAT_ID                     (RDB$STAT_ID) INTEGER Nullable
MON$STAT_GROUP                  (RDB$STAT_GROUP) SMALLINT Nullable
MON$PAGE_READS                  (RDB$COUNTER) BIGINT Nullable
MON$PAGE_WRITES                 (RDB$COUNTER) BIGINT Nullable
MON$PAGE_FETCHES                (RDB$COUNTER) BIGINT Nullable
MON$PAGE_MARKS                  (RDB$COUNTER) BIGINT Nullable


SQL> show function rdb$get_context;

Function RDB$GET_CONTEXT:
Function library is system_module
Entry point is get_context
Returns  FREE_IT VARCHAR(255) CHARACTER SET NONE
Argument 1: NULL VARCHAR(80) CHARACTER SET NONE
Argument 2: NULL VARCHAR(80) CHARACTER SET NONE


SQL> show collation ascii;

ASCII, CHARACTER SET ASCII, PAD SPACE, SYSTEM
```

You will note that the `show function` command will display details of externally defined functions, and some — but not all — internal functions (specifically, it shows `RDB$GET_CONTEXT` and `RDB$SET_CONTEXT`, but not internal functions like `ABS`).

19.19. Show Tables

```
SQL> SHOW TABLEs [name];
```

This command lists the user defined tables in the database if the first form of the command is used, or displays the columns and data types or domains making up the table if the second form is used with a table name supplied as a parameter.

```
SQL> show tables;

       COUNTRY                             CUSTOMER
       DEPARTMENT                          EMPLOYEE
       EMPLOYEE_PROJECT                    JOB
       PROJECT                             PROJ_DEPT_BUDGET
       SALARY_HISTORY                      SALES


SQL> show table country;

COUNTRY                         (COUNTRYNAME) VARCHAR(15) Not Null
CURRENCY                        VARCHAR(10) Not Null
CONSTRAINT INTEG_2:
  Primary key (COUNTRY)
```

You will note that if there are comments defined for a table, this command will *not* display them. You must use the show comments command but be aware that you will then be given all comments in the database. There doesn't appear to be a method of extracting the comments for a single object, unless you query the system tables directly.

```
SQL> comment on table country is 'This table holds details about countries.';
SQL> commit;


SQL> show comments;
...
COMMENT ON TABLE COUNTRY IS This table holds details about countries.;
...

SQL> show table country;
COUNTRY                         (COUNTRYNAME) VARCHAR(15) Not Null
CURRENCY                        VARCHAR(10) Not Null
CONSTRAINT INTEG_2:
  Primary key (COUNTRY)

SQL> select rdb$description
CON> from rdb$relations
CON> where rdb$relation_name = 'COUNTRY';

  RDB$DESCRIPTION
=================
            6:1e7
================================================================================
RDB$DESCRIPTION:
This is a table holding details about countries.
================================================================================
```

The output from the final query above is not ideal, but at least it's much less displayed information when there are lots of comments in your database.

19.20. Show Triggers

```
SQL> SHOW TRIGgers [name];
```

This command allows a list of all triggers created in the current database, to be displayed. The second form of the command allows the details and source code to be shown for a specific trigger. See also the `show procedures` and `show functions` commands.

```
SQL> show triggers;

Table name                      Trigger name                    Invalid
=============================== =============================== =======
CUSTOMER                        SET_CUST_NO
EMPLOYEE                        SAVE_SALARY_CHANGE
EMPLOYEE                        SET_EMP_NO
SALES                           POST_NEW_ORDER


SQL> show trigger set_cust_no;

Triggers on Table CUSTOMER:
SET_CUST_NO, Sequence: 0, Type: BEFORE INSERT, Active
AS
BEGIN
    if (new.cust_no is null) then
    new.cust_no = gen_id(cust_no_gen, 1);
END
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

19.21. Show Version

```
SQL> SHOW VERsion;
```

This command displays details about the Firebird software, your database and the on disc structure (ODS) in use.

```
SQL> show version;

ISQL Version: LI-V2.5.1.26351 Firebird 2.5
Server version:
Firebird/linux AMD64 (access method),
version "LI-V2.5.1.26351 Firebird 2.5"
Firebird/linux AMD64 (remote server),
version "LI-V2.5.1.26351 Firebird 2.5/tcp (hubble)/P12"
Firebird/linux AMD64 (remote interface),
version "LI-V2.5.1.26351 Firebird 2.5/tcp (hubble)/P12"
on disk structure version 11.2
```

The above output has been adjusted to fit on the page. Each pair of lines beginning with 'Firebird' and 'version' are normally displayed as a single line. They are split over two lines here.

19.22. Show Users

```
SQL> SHOW USERS;
```

This command shows a list of users who are currently connected to the database. If a user is logged in on more than one session, all sessions will be displayed separately.

```
SQL> show users;

Users in the database
# SYSDBA                              # SYSDBA
# NORMAN
```

19.23. Show Views

```
SQL> SHOW VIEWs [name];
```

The first form of this command displays a list of all views in the current database. Drilling down using the second form of the command will display the columns and source code for a specific view.

```
SQL> show views;

       PHONE_LIST


SQL> show view phone_list;

EMP_NO                          (EMPNO) SMALLINT Not Null
FIRST_NAME                      (FIRSTNAME) VARCHAR(15) Not Null
LAST_NAME                       (LASTNAME) VARCHAR(20) Not Null
PHONE_EXT                       VARCHAR(4) Nullable
LOCATION                        VARCHAR(15) Nullable
PHONE_NO                        (PHONENUMBER) VARCHAR(20) Nullable
View Source:
==== ======
 SELECT
    emp_no, first_name, last_name, phone_ext, location, phone_no
    FROM employee, department
    WHERE employee.dept_no = department.dept_no
```

Appendix A: Document History

The exact file history is recorded in the firebird-documentation git repository; see https://github.com/FirebirdSQL/firebird-documentation

Revision History

0.1	Dec 2006	KV	First version by Kamala Vadlamani.
0.2	5 Jul 2008	PV	Changed title to *Isql - Firebird Interactive SQL Utility* to bring it in line with the other manuals. Added `titleabbrev` and edition info. Moved *Audience* and *Version* sections into *Introduction*. Removed *Related Documentation* section. Fixed typos, interpunction (still more to do here). Replaced most `emphasises` and all `citetitles` with more appropriate tags. Gave IDs to manual and all (sub)sections. Added manual History and License Notice.
0.3	20 Oct 2009	ND	Converted from a chapter in the *Command Line Utilities* manual to stand alone manual in its own right. Changed title to *Firebird Interactive SQL Utility* to bring it in line with the other utility manuals. Many other updates to bring this manual into line with the others and to incorporate Firebird 2 changes etc.
0.4	15 Feb 2012	ND	General tidy up. Changes to formatting. Corrected some Docbook "misuse". Spelling & punctuation corrections. Lists compacted. Corrected <screen> overflow in pdf rendering. Etc.
0.5	10 Apr 2012	ND	More tidying up. Plus: • `Show` commands removed to a separate section. • `Set` commands moved to a separate section. • The Command Line Switches section relocated to a better place. • The section Ending an Isql Session was relocated to a better place.
0.6	20 Jun 2020	MR	Conversion to AsciiDoc, minor copy-editing

Appendix B: License Notice

The Original Documentation is titled *Firebird Interactive SQL Utility*.

The Initial Writer of the Original Documentation is: Kamala Vadlamani.

Contributor: Paul Vinkenoog - see Document history.

Contributor: Norman Dunbar - see Document history.