Firebird Interactive SQL Utility

Kamala Vadlamani

15 February 2012 – Document version 0.4

Extensive review and many changes.: Paul Vinkenoog

Firebird 2.x updates, conversion to stand alone manual.: Norman Dunbar

Table of Contents

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.

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

Audience

This manual assumes prior knowledge of basic database concepts.

Version

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

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.

> **Note**
>
> 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.

Invoking Isql

Go to the `bin` subdirectory of your Firebird installation and type **isql** (Windows) or **./isql** (Linux) at the command prompt.

Example:

```
C:\Program Files\Firebird\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';
```

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]

> **Note**
>
> 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.

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.

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'

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

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, then 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>
```

Caution

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
```

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';#
```

> **Note**
>
> 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**.

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

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.

> **Caution**
>
> 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>
```

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.

> **Note**
>
> 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

> **Note**
>
> 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;

> **Note**
>
> 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

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 from the first 127 characters of the ASCII subset, by using the **SET TERM** command.

> **Note**
>
> The default terminator maybe changed in all instances except in the case of *procedure language statements* or PSQL. PSQL does not accept any terminator other than a semicolon.

To change the terminator character to a tilda (~) 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 withing 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~
```

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 return, the command will be executed in the latter case.

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 meta data. 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

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 meta data. Meta data 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 meta data 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;
```

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

Script Handling

A batch of DDL 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 inserting, updating 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. 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.

Note

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> 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
```

Using Isql Interactively

The Firebird isql utility can be used interactively to:

- Create, update, query, and drop data and meta data.
- Add and modify data.
- Test queries.
- Perform database administrative functions.
- Input a script file containing a batch of SQL statements in sequence without prompting.

To perform these functions, isql accepts three kinds of commands at the prompt:

- DDL statements: Data Definition Language statements are used to define database schemas and/or objects. Examples of DDL commands include: **CREATE**, **ALTER**, **RECREATE** and **DROP**.
- DML statements: Data Manipulation Language statements allow the user to manipulate data objects and relationships between them, in the context of given database schemas. DML supports the manipulation and processing of database objects. Examples of DML statements are: **INSERT**, **UPDATE** and **DELETE**.
- Isql commands, which are instructions to isql itself, including **SET** and **SHOW** commands. These are discussed below.
  - Commands that perform general tasks, such as processing scripts and executing shell commands. These commands are: **INPUT**, **OUTPUT**, **QUIT**, **SHELL**, **BLOBDUMP**, **BLOBVIEW**, **HELP**, **EDIT**, **ADD**, **COPY** and **EXIT**.
    •

INPUT: reads and executes SQL scripts from a defined text file. These files can have several embedded DDL scripts in them. Scripts, can be created using a text editor or built interactively, using the **OUTPUT** or **EDIT** command. For example:

SQL> INPUT filename;

Table 3. INPUT arguments

ARGUMENTS	DESCRIPTIONS
FILENAME	Name of a file containing SQL statements and commands.

- **OUTPUT:** redirects output to a disk file or to a monitor(output screen). To output both data and commands, use **SET ECHO ON**. To output data only, use **SET ECHO OFF**. Here is the code:

  SQL> OUTPUT [filename];

  Table 4. OUTPUT arguments

ARGUMENTS	DESCRIPTIONS
FILENAME	Name of the file where output is saved. If no file name is given, results appear on the monitor.

  Example:

```
SQL> OUTPUT C:\data\managers.dta;
SQL> SELECT EMP_NO, EMP_NAME FROM MANAGER; /* Output goes to file */
SQL> OUTPUT;                               /* Output goes back to screen */
```

- **SHELL:** provides temporary access to the command line of the operating system shell without committing or rolling back any transactions. Here is the code:

  SQL> SHELL [operating system command];

  Table 5. SHELL arguments

ARGUMENTS	DESCRIPTIONS
Operating System command	A valid operating system command or call. After the command is executed, control returns to isql. In cases where no command is issued, isql opens an interactive session in the OS shell. To return control to isql, type exit.

  Example :

```
SQL> SHELL dir /mydir;
```

- **BLOBDUMP:** stores BLOB(Binary Large Object) data in a defined file. Here is the code:

SQL> BLOBDUMP blob_id filename;

Table 6. BLOBDUMP arguments

ARGUMENTS	DESCRIPTIONS
BLOB ID	Identifier consisting of two hex numbers separated by a colon (:). The first number is the ID of the table containing the BLOB column, the second is a sequenced instance number. To get the blob_id, issue any SELECT statement that selects a column of BLOB data. The output will show the hex blob_id above or in place of the BLOB column data, depending on whether SET BLOB[DISPLAY] is ON or OFF.
FILENAME	Fully qualified file system name of the file which is to receive the data.

Example :

```
SQL> BLOBDUMP 32:d48 IMAGE.JPG;
```

- **BLOBVIEW: BLOBVIEW** displays BLOB data in the default text editor. Here is the code:

  BLOBVIEW blob_id;

Table 7. BLOBVIEW arguments

ARGUMENTS	DESCRIPTIONS
BLOB ID	Identifier consisting of two hex numbers separated by a colon (:). See BLOBDUMP for instructions on how to determine the blob_id you are looking for. In current versions, BLOBVIEW does not support online editing of the BLOB. It may be introduced in a future release.
FILENAME	Fully qualified file system name of the file which is to receive the data.

Example :

```
SQL> BLOBVIEW 85:7;
```

> **Note**
>
> **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;**

- **HELP:** displays a list of isql commands with descriptions. You can combine it with **OUTPUT** to print the list to a file. Here is the code:

  SQL> HELP;

  For example, to create a text file containing all the output from the **help** command:

  ```
  SQL> OUTPUT HELPLIST.TXT;
  SQL> HELP;
  SQL> OUTPUT; /* toggles output back to the monitor */
  ```

  No arguments.
- **EDIT:** allows editing and re-execution of the previous isql command or of a batch of commands in a source file. Here is the code:

  SQL> EDIT [filename];

  Table 8. EDIT arguments

ARGUMENT	DESCRIPTION
FILENAME	Optional, fully qualified file system name of file to edit.

  Example :

  ```
  SQL> EDIT /usr/mystuff/batch.sql;
  ```
- **ADD:** adds rows interactively to a table, field after field.
- **COPY:** copies the structure of a table into a new table, either in the same database or in another.
- **EXIT:** commits the current transaction without prompting, closes the database and ends the isql session.

  > **Note**
  >
  > If you need to rollback the transaction instead of committing it, use **QUIT** instead.

  Example :

  ```
  SQL> EXIT;
  ```

  No arguments.
- **QUIT:** rolls back the current transaction without prompting, closes the database and ends the isql session.

> **Note**
>
> If you need to commit the transaction instead of rolling it back, use **EXIT** instead.

Example :

```
SQL> QUIT;
```

No arguments.

- **SHOW COMMANDS:** are used to query the database to display meta data. Meta data is stored in system tables. Meta data includes the definition of database objects such as domains, generators, tables, constraints, indices, views, triggers, stored procedures, user-defined functions(UDFs), and blob filters. **SHOW** commands run in READ COMMITTED mode to ensure the return of the most up-to-date view of the database. Here is the list of SHOW commands:
  - **SHOW DOMAIN[S]:** 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).

    Here is the syntax to display domain information:

    SQL> SHOW { DOMAINS | DOMAIN name };

    Table 9. SHOW DOMAIN[S] arguments

ARGUMENTS	DESCRIPTION
DOMAIN[S]	Lists the names of all the domains declared in the database
DOMAIN name	Displays definition of the named domain

    Example :

```
SQL> SHOW DOMAIN;

       ADDRESSLINE                             BUDGET
       COUNTRYNAME                             CUSTNO
       DEPTNO                                  EMPNO
       FIRSTNAME                               JOBCODE
...

SQL> SHOW DOMAIN ADRESSLINE;
ADDRESSLINE                     VARCHAR(30) Nullable
```

  -

SHOW GENERATOR[S]: displays information about generators. Generators are automatic sequential counters, spanning the entire database. They are outside the purview of transaction control. Here is the syntax to display generator information:

SQL> SHOW { GENERATORS | GENERATOR name };

Table 10. SHOW GENERATOR[S] arguments

ARGUMENTS	DESCRIPTION
GENERATORS	Lists the names of all generators declared in the database, along with their next values
GENERATOR NAMES	Displays the declaration of the named generator, along with its next value

Example :

```
SQL> SHOW GENERATORS;
Generator CUST_NO_GEN, current value is 1015
Generator EMP_NO_GEN, current value is 145

SQL> SHOW GENERATOR EMP_NO_GEN;
Generator EMP_NO_GEN, current value is 145
```

> **Note**
>
> The term *generator* is non standard and has been replaced by the ANSI Standard term *sequence*. The above should now be replaced by the following:

```
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
```

- **SHOW CHECK:** displays all user-defined check constraints defined for a table. Here is the syntax for the code:

SQL> SHOW CHECK <table name>;

Table 11. SHOW CHECK arguments

ARGUMENTS	DESCRIPTION
Table Name	Name of a table that exists in the attached database, and if it has any user-defined check constraints

Example :

```
SQL> SHOW CHECK COUNTRY;
There are no check constraints on table COUNTRY in this database
```

- **SHOW DATABASE:** displays information about the attached database (file name, page size and allocation, sweep interval, transaction numbers and Forced Writes status) and starting in Firebird 2.0, also reveals the On-Disk Structure or ODS version.

  On-Disk structure or ODS identifies a database with the release version of Firebird. The ODS of a database affects its compatibility with server versions. The ODS can be upgraded by using the gbak utility. In Firebird 2.0 ODS has been changed to 11.

  For version, see **SHOW VERSION**. Here is the syntax for the code:

  SQL> SHOW DATABASE | DB;

  No arguments.

  Example :

```
SQL> SHOW DATABASE;
Database: C:\Databases\Firebird\employee.fdb
        Owner: SYSDBA
PAGE_SIZE 4096
Number of DB pages allocated = 259
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 179
Transaction - oldest active = 180
Transaction - oldest snapshot = 180
Transaction - Next = 187
ODS = 11.0
Default Character set: NONE
```

- **SHOW EXCEPTION[S]:** displays exception information.

  Exceptions are user-defined named error messages, written specifically for a database and stored in that database for use in stored procedures and triggers. An exception is triggered when the value in a table is ascertained to be incorrect. This leads to a rollback of the transaction that the client application is attempting to commit. Exceptions can be interleaved. They can be 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. Here is the syntax for the code:

  SQL> SHOW { EXCEPTIONS | EXCEPTION name };

Table 12. SHOW EXCEPTION[S] arguments

ARGUMENTS	DESCRIPTION
EXCEPTIONS	Lists the names and texts of all exceptions declared in the database
EXCEPTION NAME	Displays text of the named single exception
TYPE	Stored Procedure

Example :

```
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_CHECK;

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

- **SHOW FUNCTION[S]:** displays information about user-defined functions (UDFs) declared in the attached database.

  A user-defined function (UDF) is used to perform tasks that Firebird cannot. It can be described as an external database function written entirely in another language, such as C++ or Pascal, to perform data manipulation tasks not directly supported by Firebird.

  UDFs can be called from Firebird and executed on the server. These functions can exist on their own or be collected into libraries. UDFs offer the possibility to create your own functions (such as SUBSTR) and integrate them in the database itself. Each UDF is arranged as a function, belonging to a DLL (Linux: .so). Thus one dynamically loaded library consists of at least one function.

  UDF definitions are database dependent and not server dependent, i.e. they need to be registered for each database individually. Since in Firebird, the libraries need to be stored in the Firebird UDF folder. Please refer to the **DECLARE EXTERNAL FUNCTION** statement for details of incorporating UDFs in Firebird.

> **Note**
>
> It is important to note that the majority of UDFs, when used in a WHERE condition, prevent indices from being used during execution.

Here is the syntax for the code:

SQL> SHOW { FUNCTIONS | FUNCTION name };

Table 13. SHOW FUNCTION[S] arguments

ARGUMENTS	DESCRIPTION
FUNCTION[S]	Lists the names of all UDFs declared in the database
FUNCTION NAME[S]	Displays the declaration of the named UDF

- **SHOW GRANT:** displays privileges and ROLE ownership information about a named object in the attached database; or displays user membership within roles.

  GRANT is the SQL statement, used to assign privileges to database users for specified database objects. Here is the syntax for the code:

  SQL> SHOW GRANT { object | role name };

  Table 14. SHOW GRANT arguments

ARGUMENTS	DESCRIPTION
OBJECT	Name of an existing table, view or procedure in the current database
ROLE NAME	Name of an existing role in the current database. Use SHOW ROLES to list all the roles defined for this database.

  Example:

```
SQL> SHOW GRANT;

...
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ TO PUBLIC WITH GRANT OPTION
GRANT EXECUTE ON PROCEDURE ALL_LANGS TO PUBLIC WITH GRANT OPTION
GRANT EXECUTE ON PROCEDURE DELETE_EMPLOYEE TO PUBLIC WITH GRANT OPTION
GRANT EXECUTE ON PROCEDURE DEPT_BUDGET TO PUBLIC WITH GRANT OPTION
GRANT EXECUTE ON PROCEDURE GET_EMP_PROJ TO PUBLIC WITH GRANT OPTION
...

SQL> SHOW GRANT ADD_EMP_PROJ;
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ TO PUBLIC WITH GRANT OPTION
```

-

SHOW INDEX | INDICES: displays information about a named index, about indices for a specified table or about indices for all tables in the attached database. Here is the syntax for the code:

SQL> SHOW {INDICES | INDEX { index | table } };

Table 15. SHOW INDEX| INDICES arguments

ARGUMENT	DESCRIPTION
INDEX	Name of an existing index in the current database
TABLE	Name of an existing table in the current database

> **Note**
>
> **SHOW IND** is an alias for either **SHOW INDEX** or **SHOW INDICES**.

Example:

```
SQL> SHOW INDEX;

RDB$PRIMARY1 UNIQUE INDEX ON COUNTRY(COUNTRY)
CUSTNAMEX INDEX ON CUSTOMER(CUSTOMER)
CUSTREGION INDEX ON CUSTOMER(COUNTRY, CITY)
RDB$FOREIGN23 INDEX ON CUSTOMER(COUNTRY)
RDB$PRIMARY22 UNIQUE INDEX ON CUSTOMER(CUST_NO)
...

SQL> SHOW INDEX CUSTNAMEX;
CUSTNAMEX INDEX ON CUSTOMER(CUSTOMER)
```

- **SHOW PROCEDURE[S]:** lists all procedures in the attached database, with their dependencies; or displays the text of the named procedure with the declarations and types (input/output) of any parameters. Here is the syntax for the code:

SQL> SHOW {PROCEDURES | PROCEDURE name };

Table 16. SHOW PROCEDURE[S] arguments

ARGUMENT	DESCRIPTION
NAME	Name of an existing stored procedure in the current database.

> **Note**
>
> **SHOW PROC** is an alias for either **SHOW PROCEDURE** or **SHOW PROCEDURES**.

Example:

```
SQL> SHOW PROCEDURE;
```

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

SQL> SHOW PROCEDURE ADD_EMP_PROJ;

Procedure text:
==============================================================================
BEGIN
        BEGIN
        INSERT INTO employee_project (emp_no, proj_id)
        VALUES (:emp_no, :proj_id);
        WHEN SQLCODE -530 DO
                EXCEPTION unknown_emp_id;
        END
        SUSPEND;
END
==============================================================================
Parameters:
EMP_NO                          INPUT SMALLINT
PROJ_ID                         INPUT CHAR(5)
```

- **SHOW ROLE[S]:** displays the names of SQL roles for the attached database. A role is a set of privileges on a set of database objects such as tables and views. Roles are assigned using a **GRANT** statement. To show user membership within roles, use **SHOW GRANT| ROLE NAME**.

  > **Note**
  >
  > There can be a chain of roles; for example, in a bank, the role *employee* may be granted to all tellers, and the role of *teller* may be granted to all managers, in addition to all privileges automatically granted to managers. Thus all actions executed by a session have all the privileges granted directly to the user, as well as all privileges granted to roles that are granted (directly or indirectly via other roles) to that user.

  Here is the syntax for the code:

  SQL> SHOW ROLES;

  No arguments.
- **SHOW SQL DIALECT:** displays the SQL dialects of the client and of the attached database, if there is one. Here is the syntax for the code:

  SQL> SHOW SQL DIALECT;

  Example:

```
SQL> SHOW SQL DIALECT;
        Client SQL dialect is set to: 3 and database SQL dialect is: 3
```

- **SHOW SYSTEM:** displays the names of system tables and system views for the attached database. Here is the syntax:

  SQL> SHOW SYS [TABLES];

No arguments.

TABLES is an optional keyword that does not affect the behavior of the command in versions up to Firebird 1.5.3(?). This changes in version 2.0 onwards.

In Firebird Version 2.0, the **SHOW SYSTEM** command shows predefined UDFs. The **SHOW <object_type>** command is designed to show user objects of that type. In versions earlier than 2.0, the **SHOW SYSTEM** command showed only system tables. Starting with 2.0, it also lists predefined system UDFs. In future releases it could also display views.

Shorthand: **SHOW SYS** is equivalent.

Example:

```
SQL> SHOW SYSTEM;

Tables:
       RDB$BACKUP_HISTORY                      RDB$CHARACTER_SETS
       RDB$CHECK_CONSTRAINTS                   RDB$COLLATIONS
       RDB$DATABASE                            RDB$DEPENDENCIES
       RDB$EXCEPTIONS                          RDB$FIELDS
       RDB$FIELD_DIMENSIONS                    RDB$FILES
...

Functions:
       RDB$GET_CONTEXT                         RDB$SET_CONTEXT
```

- **SHOW TABLES:** lists all tables or views in the database, and or displays information on specific named table[s] or view[s].

  Example:

```
SQL> SHOW TABLES;
       COUNTRY                                 CUSTOMER
       DEPARTMENT                              EMPLOYEE
       EMPLOYEE_PROJECT                        JOB
       PROJECT                                 PROJ_DEPT_BUDGET
       SALARY_HISTORY                          SALES
```

  > **Note**
  >
  > See also SHOW VIEW[s].

- **SHOW TRIGGERS:** displays all the triggers defined in the database along with the associated table name. A database trigger is procedural code, that is automatically executed in response to specific events, on a specified table, in a database. Here is the syntax for the code:

  SQL>SHOW TRIGGERS;

  Example:

```
SQL> SHOW TRIGGERS;
Table name                      Trigger name
==========                      ============
CUSTOMER                        SET_CUST_NO
EMPLOYEE                        SAVE_SALARY_CHANGE
```

```
EMPLOYEE                        SET_EMP_NO
SALES                           POST_NEW_ORDER
```

- **SHOW VERSION:** displays information about the software versions of isql and the Firebird server program, and the on-disk structure of the attached database. However, in Firebird 2.0 onwards the ODS version can also be returned using the **SHOW DATABASE** statement. Here is the syntax for the code:

  SQL> SHOW VERSION;

  No arguments.

  Shorthand: **SHOW VER** is equivalent.

  Example:

```
SQL> SHOW VERSION;
ISQL Version: WI-V2.0.0.12745 Firebird 2.0 Release Candidate 5
Server version:
Firebird/x86/Windows NT (access method), version "WI-V2.0.0.12745 -
Firebird 2.0 Release Candidate 5"
Firebird/x86/Windows NT (remote server), version "WI-V2.0.0.12745 -
Firebird 2.0 Release Candidate 5/XNet (PLAYTHING)/P10"
Firebird/x86/Windows NT (remote interface), version "WI-V2.0.0.12745 -
Firebird 2.0 Release Candidate 5/XNet (PLAYTHING)/P10"
on disk structure version 11.0
```

  > **Note**
  >
  > The output listed above has been split over more than one line to allow it to fit on a PDF pege. Hyphens show the location where certain lines have been split.

- **SHOW VIEW[s]:** lists all views, or displays information about the named view.

  Here is the syntax for the code:

  SQL> SHOW { VIEWS | VIEW name };

  Table 17. SHOW VIEW[S]

ARGUMENT	DESCRIPTION
NAME	Name of an existing view in the current database. The output contains column names and the SELECT statement that the view is based on.

  Example:

```
SQL> SHOW VIEW;
       PHONE_LIST
```

  > **Note**
  >
  > See also SHOW TABLES.

-

SET COMMANDS: enable users to view and change the isql environment. To view the current settings for the various set commands in the database, issue the following command:

SET;

```
SQL> SET;
Print statistics:        OFF
Echo commands:           OFF
List format:             OFF
Row Count:               OFF
Autocommit DDL:          ON
Access Plan:             OFF
Access Plan only:        OFF
Display BLOB type:       1
Column headings:         ON
Terminator:              ;
Time:                    OFF
Warnings:                ON
Bail on error:           OFF
```

- **SET AUTODDL:** specifies whether DDL statements are committed automatically after being executed, or committed only after an explicit **COMMIT**. Here is the syntax for the code:

  SQL> SET AUTODDL [ON | OFF]; /* default is ON */

  Table 18. SET AUTODDL arguments

ARGUMENT	DESCRIPTION
ON	Toggles automatic commit on.
OFF	Toggles automatic commit off.

  Shorthand : **SET AUTO** (with no argument) simply toggles AUTODDL on and off.
- **SET BLOBDISPLAY:** specifies both sub_type of BLOB to display and whether BLOB data should be displayed. Here is the syntax for the code:

  SQL> SET BLOBDISPLAY [n | ALL | OFF];

Table 19. SET BLOBDISPLAY arguments

ARGUMENT	DESCRIPTION
n	BLOB SUB_TYPE to display. Default: n= 1 (text)
ON	Display BLOB data of any sub_type
OFF	Toggles display of BLOB data off. The output shows only the Blob ID (two hex numbers separated by a colon (:). The first number is the ID of the table containing the BLOB column. The second is a sequenced instance number.

Shorthand: **SET BLOB** is the same.

- **SET COUNT:** toggles off/on whether to display the number of rows retrieved by queries. Here is the syntax for the code:

  SQL> SET COUNT [ON | OFF];

  Table 20. SET COUNT arguments

ARGUMENT	DESCRIPTION
ON	Toggles on display of "rows returned" message
OFF	Toggles off display of "rows returned" message (default)

- **SET ECHO:** toggles off/on whether commands are displayed before being executed. Default is ON but you might want to toggle it to OFF if sending your output to a script file. Here is the syntax for the code:

  SQL> SET ECHO [ON | OFF]; /* default is ON */

  Table 21. SET ECHO arguments

ARGUMENTS	DESCRIPTION
ON	Toggles on command echoing (default)
OFF	Toggles off command echoing

- **SET NAME[s]:** specifies the character set that is to be active in database transactions. Here is the syntax for the code:

  SQL> SET NAMES charset;

Table 22. SET NAMES arguments

ARGUMENTS	DESCRIPTION
CHARSET	Name of the active character set. Default: NONE

- **SET PLAN:** specifies whether to display the optimizer's query plan. Here is the syntax for the code:

  SQL> SET PLAN [ON | OFF];

  Table 23. SET PLAN arguments

ARGUMENT	DESCRIPTION
ON	Turns on display of the query plan. Default.
OFF	Turns off display of the query plan.

  Shortcut: omit ON | OFF and use just SET PLAN as a toggle.
- **SET PLANONLY:** specifies to use the optimizer's query plan and display just the plan, without executing the actual query. (Available in Firebird 1 and higher). Here is the syntax for the code:

  SQL> SET PLANONLY ON | OFF;

  The command works as a toggle switch. The argument is optional.
- **SET SQL DIALECT:** 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. Here is the syntax for the code:

  SQL> SET SQL DIALECT n;

  Table 24. SET SQL DIALECT arguments

ARGUMENT	DESCRIPTION
n	n = 1 for Dialect 1, 2 for Dialect 2, 3 for Dialect 3

- **SET STATS:** specifies whether to display performance statistics following the output of a query. Here is the syntax for the code:

  SQL> SET STATS [ON | OFF];

Table 25. SET STATS arguments

ARGUMENT	DESCRIPTION
ON	Turns on display of performance statistics. Displays: • Current memory available (bytes)• Change in available memory (bytes) • Maximum memory available (bytes) • Elapsed time for the operation (seconds) • CPU time for the operation (seconds) • Number of cache buffers used • Number of reads requested • Number of writes requested • Number of fetches done
OFF	Turns off display of performance statistics. Default.

Shortcut: omit ON | OFF and use just **SET STATS** as a toggle.

- **SET TERM:** specifies the character which will be used as the command or statement terminator, from the next statement forward. Here is the syntax for the code:

  SQL> SET TERM string;

  Table 26. SET TERM arguments

ARGUMENTS	DESCRIPTION
String	Character or characters which will be used as statement terminator. Default: ;

- **SET TIME:** specifies whether to display the time portion of a DATE value (Dialect 1 only). Here is the syntax for the code:

  SQL> SET TIME [ON | OFF];

  Table 27. SET TIME arguments

ARGUMENTS	DESCRIPTION
ON	Toggles on time portion display in Dialect 1 DATE value
OFF	Toggles on time portion display in Dialect 1 DATE value. Default.

- **SET TRANSACTION: To be completed.**
- **SET WARNINGS:** specifies whether warnings are to be output. A few examples for which isql issues warnings are:
  - SQL statement that cause no effect.
  - Pending database shutdown.
  - API calls that may be replaced in future versions of Firebird.

Here is the syntax for the code:

SQL> SET WARNINGS [ON | OFF];

Table 28. SET WARNINGS arguments

ARGUMENTS	DESCRIPTIONS
ON	Toggles on display of warnings if it was toggled off, or if the session was started with the -nowarnings option.
OFF	Toggles off display of warnings if it is currently toggled on.

Shorthand: **SET WNG** can be used as a substitute, as a simple on/off toggle.

- **SET HEADING[S]:** This allows users to disable the printing of column headers, when doing a **SELECT** inside isql, and having the output sent to a file, for processing at a later stage. In versions before 2.0, isql used to print all the column headers by default, and sometimes the sheer number of columns made the display in isql impractical. This has now been fixed with the **SET HEADING[s] on | off** toggle.

  > **Note**
  >
  > This switch cannot be deactivated with a command line parameter. Using **SET** will display the state of **SET HEAD**, along with other switches that can be toggled on/off in the isql shell.

- **SET BAIL:** will toggle the state between activated and deactivated. Using **SET** will display the state of the switch among many others. Even if **BAIL** is activated, it doesn't mean it will change isql. For example **isql -b -i my_fb.sql -o results.log -m -m2**behavior. An additional requirement should be met: the session should be non-interactive. A non-interactive session happens when the user calls isql in batch mode, giving it a script as input.

  > **Note**
  >
  > If the user loads isql interactively, and later executes a script with the input command; this is considered an interactive session, even though isql knows it is executing a script.

  Example:

```
Use CONNECT or CREATE DATABASE to specify a database
SQL> set bail;
SQL> input my_fb.sql; SQL> ^Z
```

- **SET SQLDA_DISPLAY ON | OFF:** The **SQLDA_DISPLAY** command shows the input SQLDA parameters of INSERTS, UPDATES AND DELETES. It reveals information on the raw SQLVARS. A SQLVAR represents a field in XSQLDA, the main structure used by the Firebird API to talk to clients, transferring data in and out of the server. This feature has been introduced in Firebird 2.0. It was previously available only in DEBUG builds.

> **Note**
>
> As of Firebird 2.0 this feature is not yet displayed by isql when you type SET; to view the settings of options.

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 wrapped in square brackets. For example, the **input** command will be shown as **in[put]** to indicate that the characters 'put' are optional.

Blobdump

Blobview

Edit

Help

The help command has been discussed above.

Input

Output

Shell

Exit

Quit

Isql Show commands

As explained in the **help** command, there are a number of individual show commands within isql. These are as follows:

Show Check

Show Collation

Show Database

Show Domain

Show Exception

Show Filter

Show Function

Show Generator/Sequence

Show Grant

Show Index

Show Procedure

Show Role

Show SQL Dialect

Show System

Show Table

Show Trigger

Show Version

Show Users

Show View

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:

```
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
```

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.

Set

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

```
SQL> set;

Print statistics:        OFF
Echo commands:           OFF
List format:             OFF
List Row Count:          OFF
```

```
Select rowcount limit:   0
Autocommit DDL:          ON
Access Plan:             OFF
Access Plan only:        OFF
Display BLOB type:       1
Column headings:         ON
Terminator:              ;
Time:                    OFF
Warnings:                ON
Bail on error:           OFF
```

Set Autoddl

set auto[ddl] [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**.

Set Bail

Set Blob

Set Count

Set Rowcount

Set Echo

Set Heading

Set List

Set List Rowcount CHECK THIS ONE!!!

Set Names

Set Plan

Set Planonly

Set SQLDA_Display

This is a hidden command which is not mentioned in the output from the **help set** command.

Set SQL Dialect

Set Stats

Set Time

Set Term

Set Transaction

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

Set Width

Command Line Switches

Command line switches are arguments that begin with a minus/hyphen (-) character. The following table tabulates all the switches that can be employed when executing isql.

Table 29. ISQL COMMAND-LINE SWITCHES

SWITCH	VERSION	DESCRIPTION
-a(ll)	1.0	extract meta data including legacy non-SQL tables
-b(ail)	1.5.3	instructs isql to bail out on error when used in non-interactive mode, and returns an error code to the operating system.
-c(ache) <num>		number of cache buffers
ch(arset) <charset>		connection charset (set names)
-d(atabase) <database>		database name to put in script creation
-e(cho)		echoes commands to find out where a script has caused errors (set echo on)
-ex(tract)		extract meta data
-f(etch_password)		fetch password from file
-i(nput) <file>		input file (set input)
-m(erge)		to merge standard error
-m2	2.0	merge diagnostic
-n(oautocommit)		no autocommit DDL (set autoddl off)
-nod(btriggers)		do not run database triggers
-now(arning)		do not show warnings
-o(utput) *file*		to send the output to a file(set output)
-p(assword) <password>		connection password
-pag(elength) <size>		page length
-q(uiet)		do not show the message "Use CONNECT..."
-r(ole) <role>		role name
-r2 <role>		role (uses quoted identifier)
-s(qldialect) <dialect>	1.0	SQL dialect (set sql dialect)
-t(erminator) <term>		command terminator (set term)
-u(ser) <user>		user name
-x		extract meta data
-z		shows program and server version

Some of the switches are explained in greater depth 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.

> **Note**
>
> 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, and 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.

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;

Isql's Help

Isql comes with the **HELP** command. This gives brief details of the commands available, and 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.

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

Glossary

Glossary

Glossary

DML — Data Manipulation Language. Commands that are used to create, change or delete *data*.

DDL	Data Definition Language. Commands that create, alter or drop database *objects* such as views, tables etc.
SQL	Structured Query Language. The language used to query the data in the database as well as the objects in the database.

Appendix A: Document History

The exact file history is recorded in the `manual` module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028. The full URL of the CVS log for this file can be found at http://firebird.cvs.sourceforge.net/viewvc/firebird/manual/src/docs/firebirddocs/fbutil_isql.xml?view=log

Revision History

0.1	December 2006	KV	First version by Kamala Vadlamani.
0.2	5 July 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 `emphasis`es and all `citetitle`s with more appropriate tags. Gave IDs to manual and all (sub)sections. Added manual History and License Notice.
0.3	20 October 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 February 2012	ND	General tidy up. Changes to formatting. Corrected some Docbook "misuse". Spelling & punctuation corrections. Lists compacted. Corrected <screen> overflow in pdf rendering. Etc.

Appendix B: License Notice