DHARMA SDK

ISQL Reference Manual

November 2004

Version 9.0

This manual provides reference material for the ISQL interactive tool provided in the Dharma SDK . It also includes a tutorial describing how to use the ISQL utility.

November 2004

Dharma Systems welcomes your comments on this document and the software it describes. Send comments to:

Documentation Comments

Dharma Systems, Inc.

Brookline Business Center.

#55, Route 13

Brookline, NH 03033

Phone: 603-732-4001

Fax: 603-732-4003

Electronic Mail: support@dharma.com

Web Page: http://www.dharma.com

Contents

Tables

ISQL Statements for Statement History Support 1-3
ISQL Statements for Query Formatting 1-4
Numeric Format Strings for the COLUMN Statement 1-19
Date-Time Format Strings for the Column Statement 1-19

Examples

Introduction

PURPOSE OF THIS GUIDE

This manual provides reference material for the ISQL interactive SQL utility provided in the Dharma SDK. It also includes a tutorial describing how to use the ISQL utility

AUDIENCE

The reader of this manual should be familiar with the concepts described in the Dharma SDK User Guide.

SYNTAX DIAGRAM CONVENTIONS

UPPERCASE	Uppercase type denotes reserved words. You must include reserved words in statements, but they can be upper or lower case.
lowercase	Lowercase type denotes either user-supplied elements or names of other syntax diagrams. User-supplied elements include names of tables, host-language variables, expressions and literals. Syntax diagrams can refer to each other by name. If a diagram is named, the name appears in lowercase type above and to the left of the diagram, followed by a double-colon (for example, privilege ::). The name of that diagram appears in lowercase in diagrams that refer to it.
{ }	Braces denote a choice among mandatory elements. They enclose a set of options, separated by vertical bars (|). You must choose at least one of the options.
[ ]	Brackets denote an optional element or a choice among optional elements.
|	Vertical bars separate a set of options.
...	A horizontal ellipsis denotes that the preceding element can optionally be repeated any number of times.
( ) , ;	Parentheses and other punctuation marks are required elements. Enter them as shown in syntax diagrams.

RELATED DOCUMENTATION

Refer to the following guides for more information:

Dharma SDK SQL Reference Manual	This manual describes syntax and semantics of SQL language statements and elements for the Dharma SDK .
Dharma SDK User Guide	This manual describes the Dharma Software Development Kit (SDK).It describes implementing JDBC, ODBC and .NET access to proprietary data and considerations for creating a release kit to distribute the completed implementation.
Dharma SDK ISQL Reference Manual	This manual provides reference material for the ISQL interactive tool provided in the Dharma SDK environment. It also includes a tutorial describing how to use the ISQL utility.
Dharma SDK ODBC Driver Guide	This manual describes Dharma ODBC SDK support for ODBC (Open Database Connectivity) interface and how to configure the Dharma SDK ODBC Driver.
Dharma SDK JDBC Driver Guide	Describes Dharma JDBC SDK support for the JDBC interface and how to configure the Dharma SDK JDBC Driver.
Dharma SDK .NET Data Provider Guide	This guide gives an overview of the .NET Data Provider. It describes how to set up and use the .NET Data Provider to access a Dharma SDK database from .NET applications.

Chapter 1

Introduction

1.1 OVERVIEW

Interactive SQL (often referred to throughout this manual as ISQL) is a utility supplied with Dharma SDK that lets you issue SQL statements directly from a terminal and see results displayed at the terminal. You can use interactive SQL to:

- Learn how SQL statements work
- Test and prototype SQL statements to be embedded in programs
- Modify an existing database with data definition statements
- Perform ad-hoc queries and generate formatted reports with special ISQL formatting statements

With few exceptions, you can issue any SQL statement in interactive SQL that can be embedded in a program, including CREATE, SELECT, and GRANT statements. Interactive SQL includes an online help facility with syntax and descriptions of the supported statements.

This chapter describes only those statements that are specific to ISQL. See the *Dharma SDK SQL Reference Manual* for detailed reference information on standard SQL statements that can be issued in other environments.

1.2 STARTING INTERACTIVE SQL

Start ISQL by issuing the isql command at the shell prompt. Dharma SDK invokes ISQL and displays the ISQL prompt:

```
$ isql sampledb


                    Dharma/isql Version 09.00.0000
            Dharma Systems Inc            (C) 1988-2004.
            Dharma Systems Pvt Ltd        (C) 1988-2004.


ISQL>
```

Issue Dharma SDK statements at the ISQL> prompt and terminate them with a semicolon. You can continue statements on multiple lines. ISQL automatically prompts for continuation lines until you terminate the statement with a semicolon.

To execute host operating system commands from the ISQL prompt, type HOST followed by the operating system command. After completion of the HOST statement, the ISQL> prompt returns. To execute SQL scripts from ISQL, type @ followed by the name of the file containing SQL statements.

To exit from interactive SQL, type EXIT or QUIT.

You can supply optional switches and arguments to the isql command.

Syntax

```
isql [-s script_file] [-u user_name] [-a password]
[connect_string]
```

Arguments

-s script_file
The name of an SQL script file that Dharma SDK executes when ISQL is invoked.

Note: If the file name has a space, such as:

test script.sql

The file name must be enclosed in doubles quotes, such as:

isql -s "test script.sql" testdb

-u user_name
The user name Dharma SDK uses to connect to the database specified in the *connect_string*. Dharma SDK verifies the user name against a corresponding password before it connects to the database. If omitted, the default value depends on the environment. (On UNIX, the value of the DH_USER environment variable specifies the default user name. If DH_USER is not set, the value of the USER environment variable specifies the default user name.)

-a password
The password Dharma SDK uses to connect to the database specified in the *connect_string*. Dharma SDK verifies the password against a corresponding user name before it connects to the database. If omitted, the default value depends on the environment. (On UNIX, the value of the DH_PASSWD environment variable specifies the default password.)

connect_string
A string that specifies which database to connect to. The *connect_string* can be a simple database name or a complete connect string. See the CONNECT statement in the Dharma SDK Reference Manual for details on how to specify a complete connect string. If omitted, the default value depends on the environment. (On UNIX, the value of the DB_NAME environment variable specifies the default connect string.)

1.3 STATEMENT HISTORY SUPPORT

ISQL provides statements to simplify the process of executing statements you already typed. ISQL implements a history mechanism similar to the one found in the *csh* (C-shell) supported by UNIX.

The following table summarizes the ISQL statements that support retrieving, modifying, and rerunning previously entered statements.

Table 1-1: ISQL Statements for Statement History Support

Statement	**Summary**
HISTORY	Displays a fixed number of statements (specified by the SET HISTORY statement) which have been entered before this statement, along with a statement number for each statement. Other statements take the statement number as an argument. See section “1.7.12“ on page 1-31 for details.
RUN [stmt_num]	Displays and executes the current statement or specified statement in the history buffer. See section “1.7.16“ on page 1-34 details.
LIST [stmt_num]	Displays the current statement or specified statement in the history buffer, and makes that statement the current statement by copying it to the end of the history list. See section “1.7.14“ on page 1-33 for details.
EDIT [stmt_num]	Edits the current statement or specified statement in the history buffer, and makes the edited statement the current statement by copying it to the end of the history list. The environment variable EDITOR can be set to the editor of choice. See section “1.7.8“ on page 1-27 for details.
SAVE filename	Saves the current statement in the history buffer to the specified file, which can be then be retrieved through the GET or START statements. See section “1.7.17“ on page 1-35 for details.
GET filename	Fetches the contents of the specified file, from the beginning of the file to the first semicolon, and appends it to the history buffer. The statement fetched by the GET can then be executed by using the RUN statement. See section “1.7.10“ on page 1-29 for details.
START filename [argument …]	Fetches and executes a statement stored in the specified file. Unlike the GET statement, START executes the statement and accepts arguments that it substitutes for parameter references in the statement stored in the file. START also appends the statement to the history buffer. See section “1.7.21“ on page 1-41 for details.

1.4 FORMATTING OUTPUT OF ISQL QUERIES

Formatting of database query results makes the output of a database query more presentable and understandable. The formatted output of an ISQL database query can be either displayed on the screen, written to a file, or spooled to a printer to produce a hardcopy of the report.

ISQL includes several statements that provide simple formatting of SQL queries. The following table summarizes the ISQL query-formatting statements.

Table 1-2: ISQL Statements for Query Formatting

Statement	Summary
DISPLAY	Displays text, variable values, and/or column values after the specified set of rows (called a break specification). See page 1-25 for details.
COMPUTE	Performs aggregate-function computations on column values for the specified set of rows, and assigns the results to a variable. DISPLAY statements can then refer to the variable to display its value. See page 1-23 for details.
BREAK	Specifies at what point ISQL processes associated DISPLAY and COMPUTE statements. BREAK statements can specify that processing occurs after a change in a column's value, after each row, after each page, or at the end of a query. DISPLAY and COMPUTE statements have no effect until you issue a BREAK statement with the same break specification. See page 1-13 for details.
DEFINE	Defines a variable and assigns a text value to it. When DISPLAY statements refer to the variable, ISQL prints the value. See page 1-24 for details.
COLUMN	Controls how ISQL displays a column's values (the FORMAT clause) and/or specifies alternative column-heading text (the HEADING clause). See page 1-18 for details.
TITLE	Specifies text and its positioning that ISQL displays before or after it processes a query. See page 1-44 for details.
CLEAR	Removes settings made by the previous DISPLAY, COMPUTE, COLUMN, BREAK, DEFINE, or TITLE statements. See page 1-16 for details.

SET LINESIZE
SET PAGESIZE
SET REPORT
SET ECHO
	Specifies various attributes that affect how ISQL displays queries and results.

The rest of this section provides an extended example that illustrates how to use the statements together to improve formatting.

All the examples use the same ISQL query. The query retrieves data on outstanding customer orders. The query joins two tables, *customers* and *orders*. The examples for the TABLE statement on page 1-42 show the columns and data types for these sample tables.

The following example shows the query and an excerpt of the results as ISQL displays them without the benefit of any query-formatting statements:

Example 1-1: Unformatted Query Display from ISQL

```
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
CUSTOMER_NAME                                      CUSTOMER_CITY
-------------                                      -------------
                                ORDER_ID  ORDER_VALUE
                                --------  -----------
Aerospace Enterprises Inc.                            Scottsdale
                                      13      3000000
Aerospace Enterprises Inc.                            Scottsdale
                                      14      1500000
Chemical Construction Inc.                            Joplin
                                      11      3000000
Chemical Construction Inc.                            Joplin
                                      12      7500000
Luxury Cars Inc.                                      North Rid-
geville
                                      21      6000000
Luxury Cars Inc.                                      North Rid-
geville
                                      20      5000000
.
.
.
```

Although this query retrieves the correct data, the formatting is inadequate:

- The display for each record wraps across two lines, primarily because of the column definitions for the text columns *customer_name* and *customer_city*. ISQL displays the full column width (50 characters for each column) even though the contents don't use the entire width.
- It's not clear that the values in the *order_value* column represent money amounts.

The next section shows how to use the COLUMN statement to address these formatting issues.

In addition, you can use DISPLAY, COMPUTE, and BREAK statements to present order summaries for each customer. Section 1.3.2 shows how to do this. Finally, you can add text that ISQL displays at the beginning and end of query results with the TITLE statement, as described in Section 1.3.3.

All of these statements are independent of the actual query. You do not need to change the query in any way to control how ISQL formats the results.

1.4.1 Formatting Column Display with the COLUMN Statement

You can specify the width of the display for character columns with the COLUMN statement's "An" format string. Specify the format string in the FORMAT clause of the COLUMN statement. You need to issue separate COLUMN statements for each column whose width you want to control in this manner.

The following example shows COLUMN statements that limit the width of the *customer_name* and *customer_city* columns, and re-issues the original query to show how they affect the results.

Example 1-2: Controlling Display Width of Character Columns

```
ISQL> COLUMN CUSTOMER_NAME FORMAT "A19"
ISQL> COLUMN CUSTOMER_CITY FORMAT "A19"
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
CUSTOMER_NAME       CUSTOMER_CITY          ORDER_ID  ORDER_VALUE
-------------       -------------          --------  -----------
Aerospace Enterpris Scottsdale                   13      3000000
Aerospace Enterpris Scottsdale                   14      1500000
Chemical Constructi Joplin                       11      3000000
Chemical Constructi Joplin                       12      7500000
Luxury Cars Inc.    North Ridgeville             21      6000000
Luxury Cars Inc.    North Ridgeville             20      5000000
.
.
.
```

Note that ISQL truncates display at the specified width. This means you should specify a value in the FORMAT clause that accommodates the widest column value that the query will display.

To improve the formatting of the *order_value* column, use the COLUMN statement's numeric format strings. Issue another COLUMN statement, this one for *order-_value*, and specify a format string using the "$", "9", and "," format-string characters:

- The format-string character 9 indicates the width of the largest number. Specify enough 9 format-string characters to accommodate the largest value in the column.
- The format-string character $ directs ISQL to precede column values with a dollar sign.
- The comma (,) format-string character inserts a comma at the specified position in the display.

For the *order_value* column, the format string "$99,999,999.99" displays values in a format that clearly indicates that the values represent money. (For a complete list of the valid numeric format characters, see “Table 1-3:“ on page 1-19.)

The following example shows the complete COLUMN statement that formats the *order_value* column. As shown by issuing the COLUMN statement without any arguments, this example retains the formatting from the COLUMN statements in the previous example.

Example 1-3: Customizing Format of Numeric Column Displays

```
ISQL> column order_value format "$99,999,999.99"
ISQL> column; -- Show all the COLUMN statements now in effect:
column CUSTOMER_NAME format "A19"  heading  "CUSTOMER_NAME"
column CUSTOMER_CITY format "A19"  heading  "CUSTOMER_CITY"
column ORDER_VALUE format "$99,999,999.99" heading  "ORDER_VALUE"
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
CUSTOMER_NAME       CUSTOMER_CITY    ORDER_ID   ORDER_VALUE
-------------       -------------    --------   -----------
Aerospace Enterpris  Scottsdale            13   $3,000,000.00
Aerospace Enterpris  Scottsdale            14   $1,500,000.00
Chemical Constructi  Joplin                11   $3,000,000.00
Chemical Constructi  Joplin                12   $7,500,000.00
Luxury Cars Inc.     North Ridgeville      21   $6,000,000.00
Luxury Cars Inc.     North Ridgeville      20   $5,000,000.00
.
.
.
```

1.4.2 Summarizing Data with the DISPLAY, COMPUTE, and BREAK Statements

Now that the query displays the rows it returns in a more acceptable format, you can use DISPLAY, COMPUTE, and BREAK statements to present order summaries for each customer.

All three statements rely on a break specification to indicate to ISQL when it should perform associated processing. There are four types of breaks you can specify:

- Column breaks are processed whenever the column associated with the break changes in value
- Row breaks are processed after display of each row returned by the query
- Page breaks are processed at the end of each page (as defined by the SET PAGESIZE statement)
- Report breaks are processed after display of all the rows returned by the query

While DISPLAY and COMPUTE statements specify what actions ISQL takes for a particular type of break, the BREAK statement itself controls which type of break is currently in effect. A consequence of this behavior is that DISPLAY and COMPUTE statements don't take effect until you issue the BREAK statement with the corresponding break specification.

Also, keep in mind that there can be only one type of break in effect at a time. This means you can format a particular query for a single type of break.

In our example, we are interested in a column break, since we want to display an order summary for each customer. In particular, we want to display the name of the customer along with the number and total value of orders for that customer. And, we want this summary displayed whenever the value in the *customer_name* column changes. In other words, we need to specify a column break on the *customer_nam*e column.

Approach this task in two steps. First, devise a DISPLAY statement to display the customer name and confirm that it is displaying correctly. Then, issue COMPUTE statements to calculate the statistics for each customer (namely, the count and sum of orders), and add DISPLAY statement to include those statistics. All of the DISPLAY, COMPUTE and BREAK statements have to specify the same break to get the desired results.

The following example shows the DISPLAY and BREAK statements that display the customer name. The COL clause in the DISPLAY statement indents the display slightly to emphasize the change in presentation.

The following example uses the column formatting from previous examples. Notice that the column formatting also affects DISPLAY statements that specify the same column.

Example 1-4: Specifying Column Breaks and Values with DISPLAY

```
ISQL> display col 5 "Summary of activity for", customer_name on
customer_name;
ISQL> break on customer_name
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
CUSTOMER_NAME       CUSTOMER_CITY           ORDER_ID   ORDER_VALUE
-------------       -------------           --------   -----------
Aerospace Enterpris Scottsdale                    13  $3,000,000.00
Aerospace Enterpris Scottsdale                    14  $1,500,000.00
     Summary of activity for Aerospace Enterpris
Chemical Constructi Joplin                        11  $3,000,000.00
Chemical Constructi Joplin                        12  $7,500,000.00
     Summary of activity for Chemical Constructi
Luxury Cars Inc.    North Ridgeville              21  $6,000,000.00
Luxury Cars Inc.    North Ridgeville              20  $5,000,000.00
     Summary of activity for Luxury Cars Inc.
.
.
```

.

Next, issue two COMPUTE statements to calculate the desired summary values.

COMPUTE statements specify an SQL aggregate function (AVG, MIN, MAX, SUM, or COUNT), a column name, a variable name, and a break specification. ISQL applies the aggregate function to all values of the column for the set of rows that corresponds to the break specification. It stores the result in the variable, which subsequent DISPLAY statements can use to display the result.

For this example, you need two separate compute statements. One calculates the number of orders (COUNT OF the *order_id* column) and the other calculates the total cost of orders (SUM OF the *order_value* column). Both specify the same break, namely, *customer_name*. The following example shows the COMPUTE statements, which store the resulting value in the variables *num_orders* and *tot_value*.

The following example also issues two more DISPLAY statements to display the variable values. As before, the DISPLAY statements must specify the *customer_name* break. They also indent their display farther to indicate the relationship with the previously issued DISPLAY.

As before, this example uses the COLUMN and DISPLAY statements from previous examples. ISQL processes DISPLAY statements in the order they were issued. Use the DISPLAY statement, without any arguments, to show the current set of DISPLAY statements in effect. Also, in the query results, notice that the column formatting specified for the *order_value* column carries over to the *tot_value* variable, which is based on *order_value*.

Example 1-5: Calculating Statistics on Column Breaks with COMPUTE

```
ISQL> compute count of order_id in num_orders on customer_name
ISQL> compute sum of order_value in tot_value on customer_name
ISQL> display col 10 "Total number of orders:", num_orders on
customer_name;
ISQL> display col 10 "Total value of orders:", tot_value on
customer_name;
ISQL> display  -- See all the DISPLAY statements currently active:
display  col 5  "Summary of activity for" ,customer_name  on
customer_name
display  col 10  "Total number of orders:" ,num_orders  on
customer_name
display  col 10  "Total value of orders:" ,tot_value  on customer_name
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
CUSTOMER_NAME       CUSTOMER_CITY           ORDER_ID    ORDER_VALUE
-------------       -------------           --------    -----------
Aerospace Enterpris Scottsdale                    13   $3,000,000.00
Aerospace Enterpris Scottsdale                    14   $1,500,000.00
     Summary of activity for Aerospace Enterpris
          Total number of orders:        2
          Total value of orders:   $4,500,000.00
```

```
Chemical Constructi Joplin                       11   $3,000,000.00
Chemical Constructi Joplin                       12   $7,500,000.00
     Summary of activity for Chemical Constructi
          Total number of orders:        2
          Total value of orders:  $10,500,000.00
Luxury Cars Inc.    North Ridgeville             21   $6,000,000.00
Luxury Cars Inc.    North Ridgeville             20   $5,000,000.00
     Summary of activity for Luxury Cars Inc.
          Total number of orders:        2
          Total value of orders:  $11,000,000.00
.
.
.
```

1.4.3 Adding Beginning and Concluding Titles with the TITLE Statement

You can add some finishing touches to the query display with the TITLE statement.

The TITLE statement lets you specify text that ISQL displays before (TITLE TOP) or after (TITLE BOTTOM) the query results.

The title can also be horizontally positioned by specifying the keywords LEFT, CENTER, or RIGHT; or by specifying the actual column number corresponding to the required positioning of the title.   Use the SKIP clause to skip lines after a top title or before a bottom title.

The following example uses two TITLE statements to display a query header and footer.

Example 1-6: Specifying a Query Header and Footer with TITLE

```
ISQL> TITLE TOP LEFT "Orders Summary" RIGHT "September 29, 1998" SKIP
2;
ISQL> SHOW LINESIZE  -- RIGHT alignment of TITLE is relative to this
value:
LINESIZE .................... : 78
ISQL> TITLE BOTTOM CENTER "End of Orders Summary Report" SKIP 2;
ISQL> select c.customer_name, c.customer_city, o.order_id,
o.order_value
   from customers c, orders o
   where o.customer_id = c.customer_id
   order by c.customer_name;
Orders Summary                                              September
29, 1998


CUSTOMER_NAME       CUSTOMER_CITY          ORDER_ID    ORDER_VALUE
-------------       -------------          --------    -----------
Aerospace Enterpris Scottsdale                   13   $3,000,000.00
Aerospace Enterpris Scottsdale                   14   $1,500,000.00
     Summary of activity for Aerospace Enterpris
          Total number of orders:        2
          Total value of orders:   $4,500,000.00
Chemical Constructi Joplin                        11   $3,000,000.00
Chemical Constructi Joplin                        12   $7,500,000.00
     Summary of activity for Chemical Constructi
          Total number of orders:       2
          Total value of orders:  $10,500,000.00
Luxury Cars Inc.    North Ridgeville              21   $6,000,000.00
Luxury Cars Inc.    North Ridgeville              20   $5,000,000.00
     Summary of activity for Luxury Cars Inc.
          Total number of orders:       2
          Total value of orders:  $11,000,000.00
.
.
.
Tower Construction  Munising                       8   $2,000,000.00
Tower Construction  Munising                      10   $6,000,000.00
Tower Construction  Munising                       9   $8,000,000.00
     Summary of activity for Tower Construction
          Total number of orders:       3
          Total value of orders:  $16,000,000.00


                     End of Orders Summary Report
23 records selected
ISQL>
```

1.5 THE HELP AND TABLE STATEMENTS

ISQL supports an on-line help facility that can be invoked by using the HELP statement. Typing HELP at the ISQL prompt will display a help file which will list the options accepted by the HELP statement. The various forms of the HELP statement are listed below:

- HELP — Displays the options that can be specified for HELP.
- HELP COMMANDS — Displays all the statements that ISQL accepts.
- HELP *command_name* — Displays help file corresponding to the specified statement.

TABLE is an ISQL statement that displays all the tables present in the database including any system tables. TABLE can be used also to display the description of a single table by explicitly giving the table name. Both forms of the TABLE statement are shown below:

```
TABLE;
TABLE table_name;
```

1.6 TRANSACTION SUPPORT

A transaction is started with the execution of the first SQL statement. A transaction is committed using the COMMIT WORK statement and rolled back using the ROLLBACK WORK statement.

If the AUTOCOMMIT option is set to ON, then ISQL treats each SQL statement as a single transaction. This prevents the user from holding locks on the database for an extended period of time. This is very critical when the user is querying an on-line database in which a transaction processing application is executing in real time.

A set of SQL statements can be executed as part of a transaction and committed using the COMMIT WORK statement. This is shown below:

```
<SQL statement>

<SQL statement>

<SQL statement>

COMMIT WORK ;
```

Instead, a transaction can also be rolled back using the ROLLBACK WORK statement as shown:

```
<SQL statement>

<SQL statement>

<SQL statement>

ROLLBACK WORK ;
```

An SQL statement starting immediately after a COMMIT WORK or ROLLBACK WORK statement starts a new transaction.

1.7 ISQL REFERENCE

This section provides reference material for statements specific to ISQL.

This section does not include descriptions of standard SQL statements or statements specific to embedded SQL. For details on the syntax and semantics of those other SQL statements, see the *Dharma SDK Reference Manual*.

1.7.1 @ (Execute)

Syntax

```
@filename
```

Description

Executes the SQL statements stored in the specified SQL script file. The statements specified in the file are not added to the history buffer.

Arguments

filename
The name of the script file.

Notes

The GET, START, and @ (execute) statements are similar in that they all read SQL script files. Both GET and START read an SQL script file and append the first statement in it to the history buffer. However, the START statement also executes the script statement and accepts arguments that it substitutes for parameter references in the script statement. The @ (execute) statement, on the other hand, executes all the statements in an SQL script file but does not add any of the statements to the history buffer. The @ statement does not support argument substitution.

Example

The following example shows a simple ISQL script file.

Example 1-7: Sample ISQL script

```
connect to demodb;
set echo on ;
create table stores (item_no integer, item_name char(20));
insert into stores values (1001,chassis);
insert into stores values (1002,chips);
select * from stores where item_no > 1001;
set echo off ;
```

To execute the above statements stored in a file named *cmdfile*, enter:

```
ISQL> @cmdfile
```

1.7.2 BREAK

Syntax

```
BREAK [ ON break_spec [ SKIP n ] ] ;
break_spec::
       { column_name [ , … ] | ROW | PAGE | REPORT }
```

Description

The BREAK statement specifies at what point ISQL processes associated DISPLAY and COMPUTE statements. DISPLAY and COMPUTE statements have no effect until you issue a BREAK statement with the same break specification.

A break can be specified on any of the following events:

- Change in the value of a column
- Selection of each row
- End of a page
- End of a report

Only one BREAK statement can be in effect at a time. When a new BREAK statement is entered, it replaces the previous BREAK statement. The BREAK statement can specify one or more columns on which the break can occur.

The BREAK statement without any clauses displays the currently-set break, if any.

Arguments

break_spec
The events that cause an SQL query to be interrupted and the execution of the associated COMPUTE and DISPLAY statements. *break_spec* can be any of the following values:

column_name	Causes a break when the value of the column specified by *column_name* changes.
ROW	Causes a break on every row selected by a SELECT statement.
PAGE	Causes a break at the end of each page. The end of a page is specified in the SET PAGESIZE statement. See section “1.7.18“ on page 1-36 for details on the SET statement.
REPORT	Causes a break at the end of a report or query.

SKIP n
The optional SKIP clause can be used to skip the specified number of lines when the specified break occurs and before processing of any associated DISPLAY statements.

Examples

The following examples illustrate how various break settings and corresponding DISPLAY statements affect the display of the same query.

```
ISQL> break
no break specified
ISQL> select customer_name from customers;  -- Default display
CUSTOMER_NAME
-------------
Sports Cars Inc.
Mighty Bulldozer Inc.
Ship Shapers Inc.
Tower Construction Inc.
Chemical Construction Inc.
```

```
Aerospace Enterprises Inc.
Medical Enterprises Inc.
Rail Builders Inc.
Luxury Cars Inc.
Office Furniture Inc.
10 records selected
ISQL> -- Set DISPLAY values for different breaks:
ISQL> display "Break on change in value of customer_name!" on
customer_name;
ISQL> display "Break on every row!" on row;
ISQL> display "Break on page (page size set to 2 lines)" on
page;
ISQL> display "Break on end of report!" on report;
ISQL> set pagesize 2
ISQL> break on customer_name
ISQL> select customer_name from customers;
CUSTOMER_NAME
-------------
Sports Cars Inc.
Break on change in value of customer_name!
Mighty Bulldozer Inc.
Break on change in value of customer_name!
Ship Shapers Inc.
Break on change in value of customer_name!
.
.
.
ISQL> break on row
ISQL> select customer_name from customers;
CUSTOMER_NAME
-------------
Sports Cars Inc.
Break on every row!
Mighty Bulldozer Inc.
Break on every row!
Ship Shapers Inc.
Break on every row!
.
.
.
ISQL> break on page
ISQL> select customer_name from customers;
CUSTOMER_NAME
-------------
Break on page (page size set to 2 lines)
CUSTOMER_NAME
-------------
Sports Cars Inc.
Break on page (page size set to 2 lines)
CUSTOMER_NAME
-------------
Mighty Bulldozer Inc.
Break on page (page size set to 2 lines)
.
.
.
ISQL> break on report
ISQL> select customer_name from customers;
CUSTOMER_NAME
-------------
Sports Cars Inc.
Mighty Bulldozer Inc.
Ship Shapers Inc.
Tower Construction Inc.
Chemical Construction Inc.
Aerospace Enterprises Inc.
Medical Enterprises Inc.
Rail Builders Inc.
Luxury Cars Inc.
Office Furniture Inc.
Break on end of report!
10 records selected
ISQL>
```

1.7.3 CLEAR

Syntax

```
CLEAR option ;
option::
        HISTORY
     |  BREAK
     |  COLUMN
```

```
          |  COMPUTE
          |  DISPLAY
          |  TITLE
```

Description

The CLEAR statement removes settings made by the ISQL statement corresponding to option.

Argument

option
Which ISQL statement's settings to clear:

- CLEAR HISTORY — Clears the ISQL statement history buffer.
- CLEAR BREAK — Clears the break set by the BREAK statement.
- CLEAR COLUMN — Clears formatting options set by any COLUMN statements in effect.
- CLEAR COMPUTE — Deletes clears all the options set by the COMPUTE statement.
- CLEAR DISPLAY — Clears the displays set by the DISPLAY statement.
- CLEAR TITLE— Clears the titles set by the TITLE statement.

Examples

The following example illustrates clearing the DISPLAY and BREAK settings.

```
ISQL> DISPLAY  -- See the DISPLAY settings currently in effect:
display  "Break on change in value of customeer_name!"  on
customer_name
display  "Break on every row!"  on row
display  "Break on page (page size set to 2 lines)"  on page
display  "Break on end of report!"  on report
ISQL> CLEAR DISPLAY
ISQL> DISPLAY
No display specified.
ISQL> BREAK
break on report skip 0
ISQL> CLEAR BREAK
ISQL> BREAK
no break specified
ISQL>
```

1.7.4 COLUMN

Syntax

```
COLUMN [ column_name
[ FORMAT " format_string " ] |  [ HEADING " heading_text " ] ] ;
```

Description

The COLUMN statement controls how ISQL displays a column's values (the FORMAT clause) and specifies alternative column-heading text (the HEADING clause).

The COLUMN statement without any arguments displays the current column specifications.

Arguments

column_name
The name of the column affected by the COLUMN statement. If the COLUMN statement includes *column_name* but omits both the FORMAT and HEADING clauses, ISQL clears any formatting and headings in effect for that column. The formatting specified for *column_name* also applies to DISPLAY statements that specify the same column.

FORMAT " format_string "
Specifies a quoted string that formats the display of column values. Valid values for format strings depend on the data type of the column.

Character	The only valid format string for character data types is of the form "An", where n specifies the width of the column display. The A character must be upper case.
Numeric	“Table 1-3:“ on page 1-19 shows valid format strings for numeric data types.
Date-time	“Table 1-4:“ on page 1-19 shows valid format strings for date-time data types. The format strings consist of keywords that SQL interprets and replaces with formatted values. Any other character in the format string are displayed as literals. The format strings are case sensitive. For instance, SQL replaces 'DAY' with all uppercase letters, but follows the case of 'Day'. Note that the SQL scalar function TO_CHAR offers comparable functionality and is not limited to SQL statements issued within ISQL. See the *Dharma SDK Reference Manual* for details on TO_CHAR.

COLUMN format strings also affect display values in DISPLAY statements that specify the same column or a COMPUTE value based on the column.

HEADING " heading_text "
Specifies an alternative heading for the column display. The default is the column name.

Format String Details

Table 1-3: Numeric Format Strings for the COLUMN Statement

Character	Example	Description
9	99999	Number of 9's specifies width. If the column value is too large to display in the specified format, ISQL displays # characters in place of the value.
0	09999	Display leading zeroes.
$	$9999	Prefix the display with '$'.
B	B9999	Display blanks if the value is zero.
,	99,999	Display a comma at position specified by the comma.
.	99,999.99	Display a decimal point at the specified position.
MI	99999MI	Display '-' after a negative value.
PR	99999PR	Display negative values between '<' and '>'.

Table 1-4: Date-Time Format Strings for the Column Statement

Character	Description
CC	The century as a 2-digit number.
YYYY	The year as a 4-digit number.
YYY	The last 3 digits of the year.
YY	The last 2 digits of the year.
Y	The last digit of the year.
Y,YYY	The year as a 4-digit number with a comma after the first digit.
Q	The quarter of the year as 1-digit number (with values 1, 2, 3, or 4).
MM	The month value as 2-digit number (in the range 01-12).
MONTH	The name of the month as a string of 9 characters ('JANUARY' to 'DECEMBER ').
MON	The first 3 characters of the name of the month (in the range 'JAN' to 'DEC').
WW	The week of year as a 2-digit number (in the range 01-52).
W	The week of month as a 1-digit number (in the range 1-5).
DDD	The day of year as a 3-digit number (in the range 001-365).

Table 1-4: Date-Time Format Strings for the Column Statement

Character	Description
CC	The century as a 2-digit number.
DD	The day of month as a 2-digit number (in the range 01-31).
D	The day of week as a 1-digit number (in the range 1-7, 1 for Sunday and 7 for Saturday).
DAY	The day of week as a 9 character string (in the range 'SUNDAY' to 'SATURDAY '.
DY	The day of week as a 3 character string (in the range 'SUN' to 'SAT').
J	The Julian day (number of days since DEC 31, 1899) as an 8 digit number.
TH	When added to a format keyword that results in a number, this format keyword ('TH') is replaced by the string 'ST', 'ND', 'RD' or 'TH' depending on the last digit of the number.
AMPM	The string 'AM' or 'PM' depending on whether time corresponds to forenoon or afternoon.
A.M.P.M.	The string 'A.M.' or 'P.M.' depending on whether time corresponds to forenoon or afternoon.
HH12	The hour value as a 2-digit number (in the range 00 to 11).
HHHH24	The hour value as a 2-digit number (in the range 00 to 23).
MI	The minute value as a 2-digit number (in the range 00 to 59).
SS	The seconds value as a 2-digit number (in the range 00 to 59).
SSSSS	The seconds from midnight as a 5-digit number (in the range 00000 to 86399).
MLS	The milliseconds value as a 3-digit number (in the range 000 to 999).

Examples

The following examples are based on a table, *orders*, with columns defined as follows:

```
ISQL> table orders
COLNAME                         NULL ?      TYPE        LENGTH
-------                         ------      ----        ------
order_id                        NOT NULL    INT              4
customer_id                                 INT              4
```

```
steel_type                                         CHAR          20
order_info                                         CHAR         200
order_weight                                       INT            4
order_value                                        INT            4
order_state                                        CHAR          20
```

ISQL displays the order_info column, at 200 characters, with lots of blank space preceding the values:

```
ISQL> select order_info from orders where order_value < 1000000
ORDER_INFO
----------




 Solid Rods 5 in. diameter



1 record selected
```

You can improve formatting by using the character format string to limit the width of the display:

```
ISQL> column ORDER_INFO format "A28"  heading  "Details"
ISQL> select order_info from orders where order_value <
1000000;
ORDER_INFO
----------
 Solid Rods 5 in. diameter
1 record selected
ISQL> -- Illustrate some options with numeric format strings.
ISQL> -- No column formatting:
ISQL> select order_value from orders where order_value <
1000000;
                        ORDER_VALUE
                        -----------
                             110000
1 record selected
ISQL> -- Format to display as money, and use different heading:
ISQL>  column order_value format "$999,999,999.99" heading
"Amount"
ISQL> select order_value from orders where order_value <
1000000;
              AMOUNT
              ------
```

```
     $110,000.00
1 record selected
```

The following examples use the single-value system table, *syscalctable*, and the *sysdate* scalar function, to illustrate some date-time formatting. The *sysdate* function returns today's date.

```
ISQL> select sysdate from syscalctable;  -- No formatting
SYSDATE
-------
05/07/1998
ISQL> column sysdate format "Day"
ISQL> select sysdate from syscalctable
SYSDATE
-------
    Thursday
1 record selected
ISQL> column sysdate format "Month"
ISQL> select sysdate from syscalctable
SYSDATE
-------
      May
1 record selected
ISQL> column sysdate format "DDth"
ISQL> select sysdate from syscalctable
SYSDATE
-------
           7th
1 record selected
```

Note: If the select-list of a query includes column titles, they override formatting specified in COLUMN statements for those columns. The following example illustrates this behavior.

```
ISQL> select fld from syscalctable; -- No formatting
         FLD
         ---
         100
1 record selected
ISQL> column fld heading "column title"  -- Specify heading in
COLUMN statement
ISQL> select fld from syscalctable;
COLUMN TITLE
------------
         100
```

```
1 record selected
ISQL> select fld "new title" from syscalctable;  -- Specify
title in select list
   NEW TITLE
   ---------
         100
1 record selected
```

1.7.5 COMPUTE

Syntax

```
COMPUTE
     [  { AVG | MAX | MIN | SUM | COUNT }
        OF column_name
        IN variable_name
        ON break_spec  ]  ;
break_spec::
     { column_name | ROW | PAGE | REPORT }
```

Description

Performs aggregate function computations on column values for the specified set of rows, and assigns the results to a variable. DISPLAY statements can then refer to the variable to display its value.

COMPUTE statements have no effect until you issue a BREAK statement with the same *break_spec*.

Issuing the COMPUTE statement without any arguments displays the currently-set COMPUTE specifications, if any.

Arguments

AVG | MAX | MIN | SUM | COUNT
The function to apply to values of *column_name*. The functions AVG, MAX, MIN, and SUM can be used only when the column is numeric. The function COUNT can be used for any column type.

column_name
The column whose value is to be computed. The column specified in *column_name* must also be included in the select list of the query. If *column_name* is not also included in the select list, it has no effect.

variable_name
Specifies the name of the variable where the computed value is stored. ISQL issues an implicit DEFINE statement for *variable_name* and assigns the variable a value of zero. During query processing, the value of *variable_name* changes as ISQL encounters the specified breaks.

break_spec
Specifies the set of rows after which ISQL processes the COMPUTE statement. A COMPUTE statement has no effect until you issue a corresponding BREAK statement. See the description of the BREAK statement on page 1-13 for details.

Examples

The following example computes the number of items ordered by each customer.

```
ISQL> break on customer_name

ISQL> display col 5 "Number of orders placed by",
customer_name, "=", n_ord on customer_name

ISQL> compute count of order_id in n_ord on customer_name;

ISQL> select c.customer_name, o.order_id from customers c,
orders o

where o.customer_id = c.customer_id;

CUSTOMER_NAME                                         ORDER_ID

-------------                                         --------

Sports Cars Inc.                                             1

Sports Cars Inc.                                             2

     Number of orders placed by Sports Cars Inc.

       =       2

Mighty Bulldozer Inc.                                        3

Mighty Bulldozer Inc.                                        4

   Number of orders placed by Mighty Bulldozer Inc.

       =       2

.

.

.
```

1.7.6 DEFINE

Syntax

```
DEFINE [ variable_name = value ] ;
```

Description

The DEFINE statement defines a variable and assigns an ASCII string value to it. When you refer to the defined variable in DISPLAY statements, ISQL prints the value.

The DEFINE statement is useful if you have scripts with many DISPLAY statements. You can change a single DEFINE statement to change the value in all of the DISPLAY statements that refer to the variable.

Issuing the DEFINE statement without any arguments displays any currently-defined variables, including those defined through the COMPUTE statement.

Arguments

variable_name
Specifies the name by which the variable can be referred to.

value
The ASCII string that is assigned to the variable. Enclose value in quotes if it contains any non-numeric values.

Example

The following example defines a variable called *nestate* and assigns the value NH to it.

```
ISQL> DEFINE nestate = "NH" ;
```

1.7.7 DISPLAY

Syntax

```
DISPLAY { [ col_position ] display_value } [ , … ] ON break_spec
;
col_position::
     { COL column_number | @ column_name }
display_value::
     { "text string" | variable | column_name }
break_spec::
     { column_name | ROW | PAGE | REPORT }
```

Description

The DISPLAY statement displays the specified text, variable value, and/or column value after the set of rows specified by *break_spec*. DISPLAY statements have no effect until you issue a BREAK statement with the same *break_spec*.

Issuing the DISPLAY statement without any arguments displays the currently-set DISPLAY specifications, if any.

Arguments

col_position
An optional argument that specifies the horizontal positioning of the associated display value. There are two forms for the argument:

COL column_number	Directly specifies the column position of the display value as an integer(1 specifies column 1, 2 specifies column 2, and so on.).
@column_name	Names a column in the select list of the SQL query. ISQL aligns the display value with the specified column.

If the DISPLAY statement omits *col_position*, ISQL positions the display value at column 1.

display_value
The value to display when the associated break occurs:

"text string"	If the display value is a text string, ISQL simply displays the text string.
variable	If the display value is a variable, ISQL displays the value of the variable when the associated break occurs. The variable argument refers to a variable named in a COMPUTE or DEFINE statement that executes before the query. If variable is undefined, ISQL ignores it.
column_name	If the display value is a column name, ISQL displays the value of the column when the associated break occurs. The column specified in *column_name* must also be included in the select list of the query. If *column_name* is not also included in the select list, it has no effect. If a COLUMN statement specifies a format for the same column, the formatting also affects the DISPLAY statement.

break_spec
Specifies the set of rows after which ISQL processes the DISPLAY statement. A DISPLAY statement has no effect until you issue a corresponding BREAK statement. See the description of the BREAK statement on page 1-13 for details of break specifications.

Examples

The following set of examples compute the number of orders placed by each customer and displays the message Number of orders placed by, followed by the customer name and the count of orders.

```
ISQL> break on customer_name

ISQL> display col 5 "Number of orders placed by",
customer_name, "=", n_ord on customer_name

ISQL> compute count of order_id in n_ord on customer_name;

ISQL> select c.customer_name, o.order_id from customers c,
orders o

where o.customer_id = c.customer_id;

CUSTOMER_NAME                                          ORDER_ID

-------------                                          --------

Sports Cars Inc.                                              1

Sports Cars Inc.                                              2

     Number of orders placed by Sports Cars Inc.

       =        2

Mighty Bulldozer Inc.                                         3

Mighty Bulldozer Inc.                                         4

   Number of orders placed by Mighty Bulldozer Inc.
```

```
       =        2
Ship Shapers Inc.                                                 5
Ship Shapers Inc.                                                 6
Ship Shapers Inc.                                                 7
   Number of orders placed by Ship Shapers Inc.
       =        3
Tower Construction Inc.                                           8
Tower Construction Inc.                                           9
Tower Construction Inc.                                          10
   Number of orders placed by Tower Construction Inc.
       =        3
```

If the select-list of a query includes column titles, they override DISPLAY statements that include variable or *column_name* display values for those columns:

```
ISQL> display col 5 "test display.  Sum of fld is", tmp on fld;
ISQL> compute sum of fld in tmp on fld;
ISQL> break on fld
ISQL>  select fld from syscalctable;  -- This works:
         FLD
         ---
         100
     test display.  Sum of fld is          100
1 record selected
ISQL> select fld "column title" from syscalctable;  -- DISPLAY
is disabled:
COLUMN TITLE
------------
         100
1 record selected
```

1.7.8 EDIT

Syntax

```
E[DIT] [stmt_num];
```

Description

The EDIT statement invokes a text editor to edit the specified statement from the statement history buffer. If the statement number is not specified, the last statement in the history buffer is edited. When you exit the editor, ISQL writes the buffer contents as the last statement in the history buffer.

By default, ISQL invokes the vi editor on UNIX and the MS-DOS editor on NT. You can change the default by setting the EDITOR environment variable:

- On UNIX, set the environment variable at the operating system command level:

  ```
  setenv EDITOR /usr/local/bin/gmacs
  ```

- On NT, set the environment variable in the initialization file DHSQL.INI in the windows directory:

  ```
  EDITOR = c:\msoffice\winword.exe
  ```

Examples

The following example uses the ! (shell) command to show the currently-set value of the EDITOR environment variable in the UNIX environment (it shows that it is set to invoke the GNU emacs editor). Then, the example uses the EDIT command to read in the fifth statement in the history buffer into an editing buffer.

```
ISQL> ! printenv EDITOR
/usr/local/bin/gmacs
ISQL> EDIT 5;
The following example edits the last statement in the history
buffer:
ISQL> select * from systable;  -- bad table name!
              *
error(-20005): Table/View/Synonym not found
ISQL> EDIT  -- invoke an editor to correct the error
.
.
.
ISQL> list  –- corrected statement is now the current statement:
select * from systables
ISQL> run  -- run the corrected statement
.
.
.
```

1.7.9 EXIT or QUIT

Syntax

```
EXIT
```

Description

The EXIT statement terminates the ISQL session.

Related Statements

QUIT and EXIT are synonymous. There is no difference in their effect.

1.7.10 GET

Syntax

```
G[ET] filename;
```

Description

The GET statement reads the first SQL statement stored in the specified script file.

Arguments

filename
The name of the script file. ISQL reads the file until it encounters a semicolon (;) statement terminator. It appends the statement to the history buffer as the most-recent statement.

Notes

- Execute the statement read by GET using the RUN statement.
- The GET, START, and @ (execute) statements are similar in that they all read SQL script files. Both GET and START read an SQL script file and append the first statement in it to the history buffer. However, the START statement also executes the script statement and accepts arguments that it substitutes for parameter references in the script statement. The @ (execute) statement, on the other hand, executes all the statements in an SQL script file but does not add any of the statements to the history buffer. The @ statement does not support argument substitution.

Example

Once you refine a query to return the results you need, you can store it in an SQL script file. For example, the file query.sql contains a complex query that joins several tables in a sample database.

Use the GET and RUN statements to read and execute the first statement in query.sql:

```
ISQL> GET query.sql
SELECT customers.customer_name,
         orders.order_info,
         orders.order_state,
         lot_staging.lot_location,
         lot_staging.start_date
    FROM customers,
         orders,
         lots,
         lot_staging
   WHERE ( customers.customer_id = orders.customer_id ) and
         ( lots.lot_id = lot_staging.lot_id ) and
         ( orders.order_id = lots.order_id ) and
         ( ( customers.customer_name = 'Ship Shapers Inc.' ) AND
         ( lot_staging.start_date is not NULL ) AND
         ( lot_staging.end_date is NULL ) )
ISQL> RUN
```

```
SELECT customers.customer_name,
         orders.order_info,
         orders.order_state,
         lot_staging.lot_location,
         lot_staging.start_date
    FROM customers,
         orders,
     lots,
         lot_staging
   WHERE ( customers.customer_id = orders.customer_id ) and
         ( lots.lot_id = lot_staging.lot_id ) and
         ( orders.order_id = lots.order_id ) and
         ( ( customers.customer_name = 'Ship Shapers Inc.' ) AND
         ( lot_staging.start_date is not NULL ) AND
         ( lot_staging.end_date is NULL ) )
CUSTOMER_NAME                                        ORDER_INFO
-------------                                        ----------



                      ORDER_STATE            LOT_LOCATION           START_DATE
                      -----------            ------------           ----------


Ship Shapers Inc.                                     I Beams Size 10


                      Processing             Hot Rolling            12/26/1994

1 record selected
```

1.7.11 HELP

Syntax

```
HE[LP] {COMMANDS|CLAUSES};

HE[LP] ;
```

Description

The HELP statement displays the help information for the specified statement or clause.

Notes

- HELP COMMANDS displays a list of statements for which help text is available.
- HELP CLAUSES display a list of clauses for which help text is available.
- HELP statement with no clauses display the help text for the HELP statement.

Example

The following HELP statement will give a brief description of the SELECT statement.

```
ISQL> HELP SELECT;
```

1.7.12 HISTORY

Syntax

```
HI[STORY];
```

Description

The HISTORY statement lists the statements in the statement history buffer, along with an identifying number.

Notes

- ISQL maintains a list of statements typed by the user in the statement history buffer. The SET HISTORY statement sets the size of the history buffer.
- The statements LIST, EDIT, HISTORY, and RUN are not added to the history buffer.
- Use HISTORY to obtain the statement number for a particular statement in the history buffer that you want to execute. Then, use the RUN statement with the statement number as an argument to execute that statement. Or, use LIST statement with the statement number as an argument to make the statement the current statement, which can then be executed using RUN without an argument.

Example

The following example illustrates usage of the HISTORY statement.

```
ISQL> HISTORY  -- Display statements in the history buffer
      1  start start_ex.sql Ship
      2  SELECT customer_name FROM customers
         WHERE customer_name LIKE 'Ship%'
      3  select tbl from systables where tbltype = 'T'
ISQL> RUN 2  -- Run the query corresponding to statement 2
SELECT customer_name FROM customers
WHERE customer_name LIKE 'Ship%'
CUSTOMER_NAME
-------------
Ship Shapers Inc.
1 record selected
ISQL> HI  -- In addition to executing, statement 2 is now the
current statement
      1  start start_ex.sql Ship
      2  SELECT customer_name FROM customers
```

```
          WHERE customer_name LIKE 'Ship%'
       3  select tbl from systables where tbltype = 'T'
       4  SELECT customer_name FROM customers
          WHERE customer_name LIKE 'Ship%'
ISQL> LIST 3 – Display statement 3 and copy it to the end of the
history list
select tbl from systables where tbltype = 'T'
ISQL> history  -- Statement 3 is now also the current statement
       1  start start_ex.sql Ship
       2  SELECT customer_name FROM customers
          WHERE customer_name LIKE 'Ship%'
       3  select tbl from systables where tbltype = 'T'
       4  SELECT customer_name FROM customers
          WHERE customer_name LIKE 'Ship%'
       5  select tbl from systables where tbltype = 'T'
```

1.7.13 HOST or SH or !

Syntax

```
{ HOST | SH | ! } [host_command];
```

Description

The HOST statement executes a host operating system command without terminating the current ISQL session.

Arguments

HOST | SH | !
Synonyms for directing ISQL to execute an operating system command.

host_command
The operating system command to execute. If *host_command* is not specified, ISQL spawns a subshell from which you can issue multiple operating system commands. Use the exit command to return to the ISQL> prompt.

Example

Consider a file in the local directory named query.sql. It contains a complex query that joins several tables in a sample database. From within ISQL You can display the contents of the file with the ISQL ! (shell) statement:

```
ISQL> -- Check the syntax for the UNIX 'more' command:
ISQL> host more
Usage: more [-dfln] [+linenum | +/pattern] name1 name2 ...
ISQL> -- Use 'more' to display the query.sql script file:
ISQL> ! more query.sql
```

```
SELECT customers.customer_name,
         orders.order_info,
         orders.order_state,
         lot_staging.lot_location,
         lot_staging.start_date
    FROM customers,
         orders,
         lots,
         lot_staging
   WHERE ( customers.customer_id = orders.customer_id ) and
         ( lots.lot_id = lot_staging.lot_id ) and
         ( orders.order_id = lots.order_id ) and
       ( ( customers.customer_name = 'Ship Shapers Inc.' ) AND
         ( lot_staging.start_date is not NULL ) AND
         ( lot_staging.end_date is NULL ) )    ;
ISQL> -- Spawn a subshell process to issue multiple OS commands:
ISQL> sh
.
.
.
```

1.7.14 LIST

Syntax

```
L[IST] [ stmt_num ];
```

Description

The LIST statement displays the statement with the specified statement number from the statement history buffer and makes it the current statement by adding it to the end of the history list.

If LIST omits *stmt_num*, it displays the last statement in the history buffer.

Example

The following example uses the LIST statement to display the 5th statement in the history buffer (select *customer_name* from customers) and copy it to the end of the history list. It then executes the now-current statement using the RUN statement:

```
ISQL> history
      1  title
      2  title top "fred" skip 5
      3  title
      4  help title
      5  select customer_name from customers
```

```
     6  display "Display on page break!"
     7  display "Test page break display" on page
     8  select customer_name from customers
     9  select customer_name from customers
    10  clear title
ISQL> list 5
select customer_name from customers
ISQL> run
select customer_name from customers
CUSTOMER_NAME
-------------
Sports Cars Inc.
Mighty Bulldozer Inc.
Ship Shapers Inc.
Tower Construction Inc.
Chemical Construction Inc.
Aerospace Enterprises Inc.
Medical Enterprises Inc.
Rail Builders Inc.
Luxury Cars Inc.
Office Furniture Inc.
10 records selected
ISQL>
```

1.7.15 QUIT or EXIT

Syntax

```
Q[UIT]
```

Description

The QUIT statement terminates the current ISQL session.

Related Statements

QUIT and EXIT are synonymous. There is no difference in their effect.

1.7.16 RUN

Syntax

```
R[UN] [stmt_num];
```

Description

The RUN statement executes the statement with the specified statement number from the statement history buffer and makes it the current statement by adding it to the end of the history list.

If LIST omits *stmt_num*, it runs the current statement.

Example

The following example runs the fifth statement in the history buffer.

```
ISQL> HISTORY
      1  title
      2  title top "TEST TITLE" skip 5
      3  title
      4  help title
      5  select customer_name from customers
      6  display "Display on page break!"
      7  display "Test page break display" on page
ISQL> RUN 5
select customer_name from customers
CUSTOMER_NAME
-------------
Sports Cars Inc.
Mighty Bulldozer Inc.
Ship Shapers Inc.
Tower Construction Inc.
Chemical Construction Inc.
Aerospace Enterprises Inc.
Medical Enterprises Inc.
Rail Builders Inc.
Luxury Cars Inc.
Office Furniture Inc.
10 records selected
ISQL>
```

1.7.17 SAVE

Syntax

```
S[AVE] filename;
```

Description

The SAVE statement saves the last statement in the history buffer in filename. The GET and START statements can then be used to read and execute the statement from a file.

If filename does not exist, ISQL creates it. If filename does exist, ISQL overwrites it with the contents of the last statement in the history buffer.

Example

```
ISQL> ! more test.SQL
test.sql: No such file or directory
ISQL> select customer_name, customer_city from customers;
CUSTOMER_NAME                                CUSTOMER_CITY
-------------                                -------------

Sports Cars Inc.                              Sewickley
Mighty Bulldozer Inc.                         Baldwin Park
Ship Shapers Inc.                             South Miami
Tower Construction Inc.                       Munising
Chemical Construction Inc.                    Joplin
Aerospace Enterprises Inc.                    Scottsdale
Medical Enterprises Inc.                      Denver
Rail Builders Inc.                            Claymont
.
.
.
ISQL> save test.sql
ISQL> ! ls -al test.sql
-rw-r--r--  1 dharma         51 May  1 18:21 test.sql
ISQL> ! more test.sql
select customer_name, customer_city from customers
ISQL>
```

1.7.18 SET

Syntax

```
SET set_option ;
set_option ::
          HISTORY number_statements
     |    PAGESIZE number_lines
     |    LINESIZE number_characters
     |    COMMAND LINES  number_lines
|    REPORT { ON | OFF }
|    ECHO { ON | OFF }
|    PAUSE { ON | OFF }
|    TIME { ON | OFF }
|    DISPLAY COST { ON | OFF }
|    AUTOCOMMIT { ON | OFF }
|    TRANSACTION ISOLATION LEVEL isolation_level
|    CONNECTION { database_name | DEFAULT }
```

Description

The SET statement changes various characteristics of an interactive SQL session.

Arguments

HISTORY
Sets the number of statements that ISQL will store in the history buffer. The default, and maximum, is 250 statements.

PAGESIZE number_lines
Sets the number of lines per page. The default is 72 lines. After each *number_lines* lines, ISQL executes any DISPLAY ON PAGE statements in effect and re-displays column headings. The PAGESIZE setting affects both standard output and the file opened through the SPOOL statement.

LINESIZE
Sets the number of characters per line. The default is 78 characters. The LINESIZE setting affects both standard output and the file opened through the SPOOL statement.

COMMAND LINES
Sets the number of lines to be displayed. The default is 20.

REPORT ON | OFF
SET REPORT ON copies only the results of SQL statements to the file opened by the SPOOL filename ON statement. SET REPORT OFF copies both the SQL statement and the results to the file. SET REPORT OFF is the default.

ECHO ON | OFF
SET ECHO ON displays SQL statements as well as results to standard output. SET ECHO OFF suppresses the display of SQL statements, so that only results are displayed. SET ECHO OFF is the default.

PAUSE ON | OFF
SET PAUSE ON prompts the user after displaying one page of results on the screen. SET PAUSE ON is the default.

TIME ON | OFF
SET TIME ON displays the time taken for executing a database query statement. SET TIME OFF disables the display and is the default.

DISPLAY COST ON | OFF
SET DISPLAY COST ON displays the values the Dharma SDK optimizer uses to calculate the least-costly query strategy for a particular SQL statement.

The UPDATE STATISTICS statement updates the values displayed by SET DISPLAY COST ON. SET DISPLAY COST OFF suppresses the display and is the default.

AUTOCOMMIT ON | OFF
SET AUTOCOMMIT ON commits changes and starts a new transaction immediately after each SQL statement is executed. SET AUTOCOMMIT ON is the default. SET AUTOCOMMIT OFF requires that you end transactions explicitly with a COMMIT or ROLLBACK WORK statement.

TRANSACTION ISOLATION LEVEL isolation_level
Specifies the isolation level. Isolation levels specify the degree to which one transaction can modify data or database objects being used by another concurrent transaction. See the SET TRANSACTION ISOLATION LEVEL statement in the *Dharma SDK Reference Manual* for more information on isolation levels.

CONNECTION { database_name | DEFAULT}
Sets the active connection to *database_name* or to the default connection. See the description of the CONNECT statement in the *Dharma SDK Reference Manual* for details on connections.

Notes

SET REPORT and SET ECHO are similar:

- SET REPORT affects the SPOOL file only, and ON suppresses statement display
- SET ECHO affects standard output only, and OFF suppresses statement display

Other statements control other characteristics of an interactive SQL session:

- The editor invoked by the EDIT statement is controlled by the value of the environment variable EDITOR.
- The file to which interactive SQL writes output is controlled by the SPOOL filename ON statement.

Examples

```
ISQL> -- Illustrate PAGESIZE
ISQL> DISPLAY "Here's a page break!" ON PAGE
ISQL> SET PAGESIZE 4
ISQL> BREAK ON PAGE;
ISQL> SELECT TBL FROM SYSTABLES;
TBL
---
sys_chk_constrs
Here's a page break!
```

```
TBL
---
sys_chkcol_usage
sys_keycol_usage
Here's a page break!
.
.
.
ISQL> SET DISPLAY COST ON
ISQL> -- Select from the one-record SYSCALCTABLE table:
ISQL> SELECT * FROM SYSCALCTABLE;

 Estimated Cost Values :
 -----------------------
 COST        : 8080
 CARDINALITY : 200
 TREE SIZE   : 3072

         FLD
         ---
         100
```

1.7.19 SHOW

Syntax

SHOW [show_option | SPOOL] ;

show_option ::

```
         HISTORY
     |   PAGESIZE
     |   LINESIZE
     |   COMMAND LINES
     |   REPORT
     |   ECHO
     |   PAUSE
     |   TIME
     |   DISPLAY COST
     |   AUTOCOMMIT
     |   TRANSACTION ISOLATION LEVEL
```

| CONNECTION

Description

The SHOW statement displays the values of the various settings controlled by corresponding SET and SPOOL statements. If the SHOW statement omits *show_option*, it displays all the ISQL settings currently in effect.

See the SET (page 1-36), SPOOL (page 1-40), and EDIT (page 1-27) statements for details on the settings displayed by the SHOW statement.

Example

```
ISQL> SHOW

                          ISQL  ENVIRONMENT
                        ____________________

EDITOR ....................... : vi
HISTORY buffer size .......... : 50       PAUSE ...................... : ON
COMMAND LINES ................ : 10       TIMEing command execution ... : OFF

SPOOLing ..................... : ON       LINESIZE ................... : 78
REPORTing Facility ........... : ON       PAGESIZE ................... : 72
Spool File ................... : spool_file

AUTOCOMMIT ................... : OFF      ECHO commands .............. : ON
TRANSACTION ISOLATION LEVEL .. : 0 (Snapshot)
                             DATABASE CONNECTIONS
                           ______________________

    DATABASE       CONNECTION NAME          IS DEFAULT ?      IS CURRENT ?
    --------       ---------------          ------------      ------------
    salesdb        conn_1                   No                Yes
```

1.7.20 SPOOL

Syntax

```
SPOOL filename [ON] ;
SPOOL OFF ;
SPOOL OUT ;
```

Description

The SPOOL statement writes output from interactive SQL statements to the specified file.

Arguments

filename ON
Opens the file specified by filename and writes the displayed output into that file. The filename cannot include punctuation marks such as a period (.) or comma (,).

OFF
Closes the file opened by the SPOOL ON statement.

OUT
Closes the file opened by the SPOOL ON statement and prints the file. The SPOOL OUT statement passes the file to the system utility statement pr and the output is piped to *lpr*.

Example

To record the displayed output into the file called STK, enter:

```
ISQL> SPOOL STK ON ;

ISQL> SELECT * FROM customer ;

ISQL> SPOOL OFF ;
```

1.7.21 START

Syntax

```
ST[ART] filename [ argument ] [ ... ] ;
```

Description

The START statement executes the first SQL statement stored in the specified script file.

Arguments

filename
The name of the script file. ISQL reads the file until it encounters a semicolon (;) statement terminator.

argument …
ISQL substitutes the value of argument for parameter references in the script. Parameter references in a script are of the form &n, where n is an integer. ISQL replaces all occurrences of &1 in the script with the first argument value, all occurrences of &2 with the second argument value, and so on. The value of argument must not contain spaces or other special characters.

Notes

- In addition to executing the first statement in the script file, the START statement appends the statement (after any argument substitution) to the history buffer.
- The GET, START, and @ (execute) statements are similar in that they all read SQL script files. Both GET and START read an SQL script file and append the first statement in it to the history buffer. However, the START statement also executes the script statement and accepts arguments that it substitutes for parameter references in the script statement. The @ (execute) statement, on the other

hand, executes all the statements in an SQL script file but does not add any of the statements to the history buffer. The @ statement does not support argument substitution.

Example

```
ISQL> -- Nothing in history buffer:
ISQL> history
History queue is empty.
ISQL> -- Display a script file with the ! shell statement.  The
script's SQL ISQL> -- statement uses the LIKE predicate to retrieve
customer names
ISQL> -- beginning with the string passed as an argument in a START
statement:
ISQL> ! more start_ex.sql
SELECT customer_name FROM customers
WHERE customer_name LIKE '&1%';
ISQL> -- Use the START statement to execute the SQL statement in the
script
ISQL> -- start_ex.sql.  Supply the value 'Ship' as a substitution argu-
ment:
ISQL> START start_ex.sql Ship
CUSTOMER_NAME
-------------
Ship Shapers Inc.
1 record selected
ISQL> -- ISQL puts the script statement, after argument substitution,
ISQL> -- in the history buffer:
ISQL> history
      1  ! more start_ex.sql
      3  START start_ex.sql Ship
      4  SELECT customer_name FROM customers
         WHERE customer_name LIKE 'Ship%'
```

1.7.22 TABLE

Syntax

```
T[ABLE] [ tablename ] ;
```

Description

The TABLE statement with no argument displays a list of all the user tables in the database that are owned by the current user.

With the *tablename* argument, the TABLE statement displays a brief description of the columns in the specified table.

Examples

You can use the TABLE statement to see the structure of system tables. Unless you are logged in as the Dharma database administrator (the user *dharma*, by default), you need to qualify the system table name with the administrator user name, as in the following example:

```
ISQL> table dharma.systables
COLNAME                          NULL ?      TYPE        LENGTH
-------                          ------      ----        ------
id                               NOT NULL    INT              4
tbl                              NOT NULL    VARCHAR         32
creator                          NOT NULL    VARCHAR         32
owner                            NOT NULL    VARCHAR         32
tbltype                          NOT NULL    VARCHAR          1
tblpctfree                       NOT NULL    INT              4
segid                            NOT NULL    INT              4
has_pcnstrs                      NOT NULL    VARCHAR          1
has_fcnstrs                      NOT NULL    VARCHAR          1
has_ccnstrs                      NOT NULL    VARCHAR          1
has_ucnstrs                      NOT NULL    VARCHAR          1
tbl_status                       NOT NULL    VARCHAR          1
rssid                            NOT NULL    INT              4
```

The following example uses the *table* command to detail the structure of the tables used in examples throughout this chapter.

```
ISQL> table  - List the sample tables
TABLENAME
---------
customers
lot_staging
lots
orders
quality
samples
ISQL> table customers
COLNAME                          NULL ?      TYPE        LENGTH
-------                          ------      ----        ------
customer_id                      NOT NULL    INT              4
customer_name                                CHAR            50
customer_street                              CHAR           100
customer_city                                CHAR            50
customer_state                               CHAR            10
customer_zip                                 CHAR             5
ISQL> table orders
COLNAME                          NULL ?      TYPE        LENGTH
-------                          ------      ----        ------
order_id                         NOT NULL    INT              4
customer_id                                  INT              4
steel_type                                   CHAR            20
order_info                                   CHAR           200
```

```
order_weight                                INT             4
order_value                                 INT             4
order_state                                 CHAR           20
ISQL> table lots
COLNAME                          NULL ?     TYPE       LENGTH
-------                          ------     ----       ------
lot_id                           NOT NULL   INT             4
order_id                         NOT NULL   INT             4
lot_units                                   INT             4
lot_info                                    CHAR          200
ISQL> table lot_staging
COLNAME                          NULL ?     TYPE       LENGTH
-------                          ------     ----       ------
lot_id                                      INT             4
lot_location                                CHAR           20
start_date                                  DATE
end_date                                    DATE
issues                                      CHAR          200
ISQL> table quality
COLNAME                          NULL ?     TYPE       LENGTH
-------                          ------     ----       ------
lot_id                           NOT NULL   INT             4
purity                                      DOUBLE          8
p_deviation                                 DOUBLE          8
strength                                    DOUBLE          8
s_deviation                                 DOUBLE          8
comments                                    CHAR          200
ISQL> table samples
COLNAME                          NULL ?     TYPE       LENGTH
-------                          ------     ----       ------
lot_id                                      INT             4
samples                                     INT             4
comments                                    CHAR          200
ISQL>
```

1.7.23 TITLE

Syntax

```
TITLE    [
          [ TOP | BOTTOM ]
          [ [ LEFT | CENTER | RIGHT | COL n ] " text " ] [ … ]
          [ SKIP n ]
          ] ;
```

Description

The TITLE statement specifies text that ISQL displays either before or after it processes a query. TITLE with no arguments displays the titles currently set, if any.

Arguments

TOP | BOTTOM
Specifies whether the title is to be printed at the top or bottom of the page. The default is TOP.

LEFT | CENTER | RIGHT | COL n
Specifies the horizontal alignment of the title text: LEFT aligns the text to the left of the display; CENTER centers the text; RIGHT aligns the text to the right (with the right-most character in the column specified by the SET LINESIZE statement). COL n displays the text starting at the specified column (specifying COL 0 is the same as LEFT).

The default is LEFT.

" text "

The text to be displayed.

SKIP n
Skips the specified number of lines after a TOP title is printed and before a BOTTOM title is printed. By default, ISQL does not skip any lines.

Examples

The following example shows the effect of specifying a top title without a bottom title, then both a top and bottom title.

```
ISQL> TITLE "fred"
ISQL> select * from syscalctable;
fred
         FLD
         ---
         100
1 record selected
ISQL> TITLE BOTTOM "flintstone"
ISQL> select * from syscalctable;
fred
         FLD
         ---
         100
flintstone
1 record selected
The TITLE statement can specify separate positions for different text
in the same title:
ISQL> CLEAR TITLE
ISQL> TITLE TOP LEFT "Align on the left!" CENTER "Centered text" RIGHT
"Right aligned text!"

ISQL> select * from syscalctable;

Align on the left!             Centered text             Right aligned
text!

         FLD

         ---

         100


1 record selected
```

Index