User Guide

Interactive SQL
Command-Line Interface
# Contents

1. **Introduction** .................................................................................................................. 1

2. **Quick Tour** ..................................................................................................................... 2

   2.1 Introductory Tutorial .................................................................................................... 3
       Init 4
       Define ............................................................................................................................... 5
       Manage ............................................................................................................................. 6
       Done ................................................................................................................................. 7
       Additional Resources .................................................................................................... 8

   2.2 Relationships .................................................................................................................. 9
       Init 11
       Define ............................................................................................................................ 12
       Manage .......................................................................................................................... 13
       Done ............................................................................................................................... 14
       Additional Resources ................................................................................................... 15

   2.3 Record/Row Locking ...................................................................................................... 16
       Init 17
       Define ............................................................................................................................ 18
       Manage .......................................................................................................................... 19
       Done ............................................................................................................................... 20
       Additional Resources ................................................................................................... 21

   2.4 Transaction Processing ................................................................................................. 22
       Init 23
       Define ............................................................................................................................ 24
       Manage .......................................................................................................................... 25
       Done ............................................................................................................................... 26
       Additional Resources ................................................................................................... 27

3. **ISQL Statements** .......................................................................................................... 28

   3.1 Starting Interactive SQL ............................................................................................... 28

   3.2 Statement History Support .......................................................................................... 29

   3.3 Formatting Output of ISQL Queries ............................................................................. 30
       Formatting Column Display with the COLUMN Statement ............................................. 30
       Summarizing Data with DISPLAY, COMPUTE, and BREAK Statements ................... 32
       Adding Beginning and Concluding Titles with the TITLE Statement ......................... 33

   3.4 The HELP and TABLE Statements .............................................................................. 36

   3.5 Transaction Support .................................................................................................... 37
1. **Introduction**

Interactive SQL (often referred to in this manual as ISQL) is a utility supplied with c-treeSQL 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, any SQL statement that can be embedded in a program can be issued in interactive SQL, including **CREATE**, **SELECT**, and **GRANT** statements. Interactive SQL includes an online help facility with syntax and descriptions of the supported statements.

**SQL Data Load Tips**

ISQL is very useful for running interactive SQL statements. As an option, it allows creating and executing scripts to automate certain operations. These scripts can be used for running DDL statements (Data Definition Language: create table, create index, alter table, etc.) and DML statements (Data Manipulation Language: insert, update, etc.). DML statements can become quite large if they must insert or update every record in a large table. ISQL is not intended for massive data load operations which require a large number of Insert statements.

FairCom provides tools that are more appropriate for manipulating large numbers of records:

- **dbload** ([http://docs.faircom.com/doc/isql/32580.htm](http://docs.faircom.com/doc/isql/32580.htm)) loads records from an input data file into tables of a database.
- **dbdump** ([http://docs.faircom.com/doc/isql/32623.htm](http://docs.faircom.com/doc/isql/32623.htm)) unload utility writes the data in a database to a file.
2. **Quick Tour**

The tutorials in this chapter will introduce the basic use of c-treeACE ISQL, the Interactive SQL Interface.
2.1 Introductory Tutorial

sdk\sql\interactive\tutorials\iSQL_Tutorial1.sql

This tutorial will take you through the basic use of the c-treeACE SQL ISQL - Interactive SQL Interface.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You’re Done()!

Tutorial #1: Introductory - Simple Single Table

We wanted to keep this program as simple as possible. This program does the following:

- Initialize() - Connects to the c-treeACE Database Engine.
- Define() - Defines and creates a "customer master" (custmast) table/file.
- Manage() - Adds a few rows/records; Reads the rows/records back from the database; displays the column/field content; and then deletes the rows/records.
- Done() - Disconnects from c-treeACE Database Engine.

Note these steps in our SQL script:

```
SET ECHO OFF
-- Initialize
ECHO INIT;

-- Define
ECHO DEFINE;

-- Manage
ECHO MANAGE;

-- Done
ECHO DONE;
```

We suggest opening the source code with your own editor.

Continue now to review these four steps.
Init

First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for Initialize():

SET ECHO OFF

-- Initialize
ECHO INIT;

SET AUTOCOMMIT ON;
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```sql
-- Define
ECHO DEFINE;

ECHO Create table...;
CREATE TABLE custmast (  
    cm_custnumb CHAR(4),  
    cm_custzipc CHAR(9),  
    cm_custstat CHAR(2),  
    cm_custrtng CHAR(1),  
    cm_custname VARCHAR(47),  
    cm_custaddr VARCHAR(47),  
    cm_custcity VARCHAR(47));
```
Manage

The manage step provides data management functionality for your application and/or process.

Below is the code for Manage():

```sql
-- Manage
ECHO MANAGE;

ECHO Delete records...;
DELETE FROM custmast;

ECHO Add records...;
INSERT INTO custmast VALUES ('1000', '92867', 'CA', '1', 'Bryan Williams', '2999 Regency', 'Orange');
INSERT INTO custmast VALUES ('1001', '61434', 'CT', '1', 'Michael Jordan', '13 Main', 'Harford');
INSERT INTO custmast VALUES ('1002', '73677', 'GA', '1', 'Joshua Brown', '4356 Cambridge', 'Atlanta');
INSERT INTO custmast VALUES ('1003', '10034', 'MO', '1', 'Keyon Dooling', '19771 Park Avenue', 'Columbia');

ECHO Display records...;
SELECT cm_custnumb "Number", cm_custname "Name" FROM custmast;

ECHO Delete records...;
DELETE FROM custmast;
```
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```
-- Done
ECHO DONE;
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in iSQL_Tutorial1.sql in your installation directory, within the 'sdk\sql.interactive\tutorials' directory for your platform. Example for the Windows platform: C:\FairCom\V\win32\sdk\sql.interactive\tutorials\iSQL_Tutorial1.sql.
- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.2 Relationships

sdk\sql\interactive\tutorials\iSQL_Tutorial2.sql

Now we will build some table/file relationships using the c-treeACE SQL ISQL - Interactive SQL Interface.

This tutorial will advance the concepts introduced in the first tutorial by expanding the number of tables. We will define key columns/fields and create specific indices for each table to form a relational model database.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You’re Done()!

Tutorial #2: Relational Model and Indexing

Here we add a bit more complexity, introducing multiple tables, with related indices in order to form a simple "relational" database simulating an Order Entry system. Here is an overview of what will be created:

Relational Model Tables

- Initialize() - Connects to the c-treeACE Database Engine.
- Define() - Defines and creates the "custmast", "custordr", "orditem" and the "itemmast" tables/files with related indices.
- Manage() - Adds some related rows/records to all tables/files. Then queries the database.
- Done() - Disconnects from c-treeACE Database Engine.

Note these steps in our SQL script:

```sql
SET ECHO OFF

-- Initialize
ECHO INIT;

-- Define
ECHO DEFINE;
```
Quick Tour

-- Manage
ECHO MANAGE;

-- Done
ECHO DONE;

We suggest opening the source code with your own editor.

Continue now to review these four steps.
**Init**

First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for `Initialize()`:

```sql
SET ECHO OFF

-- Initialize
ECHO INIT;

SET AUTOCOMMIT ON;
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```sql
-- Define
ECHO DEFINE;

ECHO Drop custmast table from Tutorial 1
DROP TABLE custmast;

ECHO table CustomerMaster...;
CREATE TABLE custmast (  
    cm_custnumb CHAR(4),  
    cm_custzipc CHAR(9),  
    cm_custstat CHAR(2),  
    cm_custrtg CHAR(1),  
    cm_custname VARCHAR(47),  
    cm_custaddr VARCHAR(47),  
    cm_custcity VARCHAR(47));
CREATE UNIQUE INDEX cm_custnumb_idx ON custmast (cm_custnumb);

ECHO table CustomerOrders...;
CREATE TABLE custordr (  
    co_ordrdate DATE,  
    co_promdate DATE,  
    co_ordrnumb CHAR(6),  
    co_custnumb CHAR(4));
CREATE UNIQUE INDEX co_ordrnumb_idx ON custordr (co_ordrnumb);
CREATE INDEX co_custnumb_idx ON custordr (co_custnumb);

ECHO table OrderItems...;
CREATE TABLE ordritem (  
    oi_sequnumb SMALLINT,  
    oi_quantiy SMALLINT,  
    oi_ordrnumb CHAR(6),  
    oi_itemnumb CHAR(5));
CREATE UNIQUE INDEX oi_ordrnumb_idx ON ordritem (oi_ordrnumb, oi_sequnumb);
CREATE INDEX oi_itemnumb_idx ON ordritem (oi_itemnumb);

ECHO table ItemMaster...;
CREATE TABLE itemmast (  
    im_itemwght INTEGER,  
    im_itempric MONEY,  
    im_itemnumb CHAR(5),  
    im_itemdesc VARCHAR(47));
CREATE UNIQUE INDEX im_itemnumb_idx ON itemmast (im_itemnumb);
```
Manage

The manage step provides data management functionality for your application and/or process.

Below is the code for Manage():

```sql
-- Manage
ECHO MANAGE;
ECHO Delete records...;
DELETE FROM custmast;
DELETE FROM custordr;
DELETE FROM ordritem;
DELETE FROM itemmast;
ECHO Add records...;
INSERT INTO custmast VALUES ('1000', '92867', 'CA', '1', 'Bryan Williams', '2999 Regency', 'Orange');
INSERT INTO custmast VALUES ('1001', '61434', 'CT', '1', 'Michael Jordan', '13 Main', 'Harford');
INSERT INTO custmast VALUES ('1002', '73677', 'GA', '1', 'Joshua Brown', '4356 Cambridge', 'Atlanta');
INSERT INTO custmast VALUES ('1003', '10034', 'MO', '1', 'Keyon Dooling', '19771 Park Avenue', 'Columbia');
INSERT INTO custordr VALUES ('09/01/2002', '09/05/2002', '1', '1001');
INSERT INTO custordr VALUES ('09/02/2002', '09/06/2002', '2', '1002');
INSERT INTO ordritem VALUES (1, 2, '1', '1');
INSERT INTO ordritem VALUES (2, 1, '1', '2');
INSERT INTO ordritem VALUES (3, 1, '1', '3');
INSERT INTO ordritem VALUES (1, 3, '2', '3');
INSERT INTO itemmast VALUES (10, 19.95, '1', 'Hammer');
INSERT INTO itemmast VALUES (3, 9.99, '2', 'Wrench');
INSERT INTO itemmast VALUES (4, 16.59, '3', 'Saw');
INSERT INTO itemmast VALUES (1, 3.98, '4', 'Pliers');
ECHO;
ECHO Query results;
SELECT cm_custname "Name", SUM(im_itempric * oi_quantity) "Total"
  FROM custmast, custordr, ordritem, itemmast
  WHERE co_custnumb = cm_custnumb AND co_ordrnumb = oi_ordrnumb AND oi_itemnumb = im_itemnumb
  GROUP BY cm_custnumb, cm_custname;
```
Done

When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```
-- Done
ECHO DONE;
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in iSQL_Tutorial2.sql in your installation directory, within the 'sdk\sql.interactive\tutorials' directory for your platform.
  Example for the Windows platform: C:\FairCom\V\win32\sdk\sql.interactive\tutorials\iSQL_Tutorial2.sql.
- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.3 Record/Row Locking

Now we will explore row/record locks using the c-treeACE SQL ISQL - Interactive SQL Interface. The functionality for this tutorial focuses on inserting/adding rows/records, then updating a single row/record in the customer master table under locking control. The application will pause after a LOCK is placed on a row/record. Another instance of this application should then be launched, which will block, waiting on the lock held by the first instance. Pressing the <Enter> key will enable the first instance to proceed. This will result in removing the lock thereby allowing the second instance to continue execution. Launching two processes provides a visual demonstration of the effects of locking and a basis for experimentation on your own.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and you're Done()!

**Tutorial #3: Locking**

Here we demonstrate the enforcement of data integrity by introducing record/row “locking”.

- **Initialize()** - Connects to the c-treeACE Database Engine.
- **Define()** - Defines and creates a "customer master" (custmast) table/file.
- **Manage()** - Adds a few rows/records; Reads the rows/records back from the database; displays the column/field content. Then demonstrates an update operation under locking control, and a scenario that shows a locking conflict.
- **Done()** - Disconnects from c-treeACE Database Engine.

Note these steps in our SQL script:

```
SET ECHO OFF
-- Initialize
ECHO INIT;
-- Define
ECHO DEFINE;
-- Manage
ECHO MANAGE;
-- Done
ECHO DONE;
```

We suggest opening the source code with your own editor.

Continue now to review these four steps.
Init

First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for `Initialize()`:

```
SET ECHO OFF

-- Initialize
ECHO INIT;

SET AUTOCOMMIT OFF;
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for Define():

```sql
-- Define
ECHO DEFINE;

ECHO Create table...;
CREATE TABLE custmast (  
    cm_custnumb CHAR(4),
    cm_custzipc CHAR(9),
    cm_custstat CHAR(2),
    cm_custrtng CHAR(1),
    cm_custname VARCHAR(47),
    cm_custaddr VARCHAR(47),
    cm_custcity VARCHAR(47));
CREATE UNIQUE INDEX cm_custnumb_idx ON custmast (cm_custnumb);
COMMIT WORK;
```
Manage

The manage step provides data management functionality for your application and/or process.

Below is the code for `Manage()`:

```sql
-- Manage
ECHO MANAGE;

ECHO Delete records...;
DELETE FROM custmast;

ECHO Add records...;
INSERT INTO custmast VALUES ('1000', '92867', 'CA', '1', 'Bryan Williams', '2999 Regency', 'Orange');
INSERT INTO custmast VALUES ('1001', '61434', 'CT', '1', 'Michael Jordan', '13 Main', 'Harford');
INSERT INTO custmast VALUES ('1002', '73677', 'GA', '1', 'Joshua Brown', '4356 Cambridge', 'Atlanta');
INSERT INTO custmast VALUES ('1003', '10034', 'MO', '1', 'Keyon Dooling', '19771 Park Avenue', 'Columbia');
COMMIT WORK;

UPDATE custmast SET cm_custname = 'KEYON DOOLING' WHERE cm_custnumb = '1003';
ECHO Issue a COMMIT WORK to commit changes and release locks
```
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```
-- Done
ECHO DONE;
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in `iSQL_Tutorial3.sql` in your installation directory, within the `sdk\sql.interactive\tutorials` directory for your platform. Example for the Windows platform:
  C:\FairCom\V*\win32\sdk\sql.interactive\tutorials\iSQL_Tutorial3.sql.
- Additional documentation may be found on the FairCom Web site at: www.faircom.com
2.4 Transaction Processing

Now we will discuss transaction processing as it relates to the c-treeACE SQL ISQL - Interactive SQL Interface.

Transaction processing provides a safe method by which multiple database operations spread across separate tables/files are guaranteed to be atomic. By atomic, we mean that, within a transaction, either all of the operations succeed or none of the operations succeed. This "either all or none" atomicity ensures that the integrity of the data in related tables/files is secure.

Like all other examples in the c-tree tutorial series, this tutorial simplifies the creation and use of a database into four simple steps: Initialize(), Define(), Manage(), and You’re Done()!

**Tutorial #4: Transaction Processing**

Here we demonstrate transaction control.

- **Initialize()** - Connects to the c-treeACE Database Engine.
- **Define()** - Defines and creates our four tables/files.
- **Manage()** - Adds rows/records to multiple tables/files under transaction control.
- **Done()** - Disconnects from c-treeACE Database Engine.

Note these steps in our SQL script:

```sql
SET ECHO OFF
-- Initialize
ECHO INIT;

-- Define
ECHO DEFINE;

-- Manage
ECHO MANAGE;

-- Done
ECHO DONE;
```

We suggest opening the source code with your own editor.

Continue now to review these four steps.
**Init**

First we need to open a connection to a database by providing the c-treeACE Database Engine with a user name, password and the database name.

Below is the code for `Initialize()`:

```
SET ECHO OFF

-- Initialize
ECHO INIT;

SET AUTOCOMMIT OFF;
```
Define

The define step is where specific data definitions are established by your application and/or process. This involves defining columns/fields and creating the tables/files with optional indices.

Below is the code for `Define()`:

```c
/*
 * Define()
 * 
 * Create the table for containing a list of existing customers
 */
```
Manage

The manage step provides data management functionality for your application and/or process.

Below is the code for `Manage()`:

```
/*
 * Manage()
 *
 * This function performs simple record functions of add, delete and gets
 */
```
When an application and/or process has completed operations with the database, it must release resources by disconnecting from the database engine.

Below is the code for `Done()`:

```c
/*
 * Done()
 *
 * This function handles the housekeeping of closing connection and
 * freeing of associated memory
 */
```
Additional Resources

We encourage you to explore the additional resources listed here:

- Complete source code for this tutorial can be found in iSQL_Tutorial4.sql in your installation directory, within the 'sdk\sql.interactive\tutorials' directory for your platform. Example for the Windows platform:
  C:\FairCom\V*\win32\sdk\sql.interactive\tutorials\iSQL_Tutorial4.sql.

- Additional documentation may be found on the FairCom Web site at: www.faircom.com
3. **ISQL Statements**

This chapter describes only those statements that are specific to c-treeACE SQL ISQL. See the *c-treeSQL Reference Guide* for detailed reference information on standard c-treeACE SQL statements that can be issued in other environments.

3.1 **Starting Interactive SQL**

Start c-treeACE SQL ISQL by issuing the `ISQL` command at the shell prompt. c-treeACE SQL invokes ISQL and displays the ISQL> prompt:

```
# isql sampledb
```

```
c-treeSQL Interactive Interpreter
```

```
ISQL>
```

Issue c-treeACE SQL 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 c-treeACE SQL ISQL prompt, type HOST followed by the operating system command. After completion of the HOST statement, the ISQL> prompt returns. To execute c-treeACE SQL scripts from c-treeACE SQL ISQL, type ‘@’ followed by the name of the file containing c-treeACE SQL statements.

To exit the c-treeACE SQL ISQL, type EXIT or QUIT.

You can supply optional switches and arguments to the c-treeACE SQL ISQL command.

**Syntax**

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

**Arguments**

* -s script_file
  The name of a c-treeACE SQL script file that c-treeACE SQL executes when it invokes c-treeACE SQL ISQL.

  **Note:** For Windows platforms, 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 c-treeACE SQL uses to connect to the database specified in the `connect_string`. c-treeACE SQL verifies a 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 c-treeSQL uses to connect to the database specified in the **connect_string**.
  c-treeSQL 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. For example, to connect to a local database named 'myDatabase', you would use the following syntax:

```bash
# isql -u ADMIN -a ADMIN myDatabase
```

To connect to a remote database named c-treeSQL, you would use the **6597@remotestore:database** syntax as follows:

```bash
# isql -u ADMIN -a ADMIN 6597@hotdog.faircom.com:ctreeSQL
```

See the **CONNECT** statement in the **c-treeSQL 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.)

### 3.2 Statement History Support

c-treeSQL ISQL provides statements to simplify the process of executing statements you have 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>
<thead>
<tr>
<th>Statement</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>HISTORY</td>
<td>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 &quot;HISTORY&quot; (page 60) for details.</td>
</tr>
<tr>
<td>RUN [ stmt_num ]</td>
<td>Displays and executes the current statement or specified statement in the history buffer. See &quot;RUN&quot; (page 66) for details.</td>
</tr>
<tr>
<td>LIST [ stmt_num ]</td>
<td>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 &quot;LIST&quot; (page 64) for details.</td>
</tr>
<tr>
<td>EDIT [ stmt_num ]</td>
<td>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 &quot;EDIT&quot; (page 55) for details.</td>
</tr>
<tr>
<td>SAVE filename</td>
<td>Saves the current statement in the history buffer to the specified file, which can then be retrieved through the GET or START statements. See &quot;SAVE&quot; (page 67) for details.</td>
</tr>
</tbody>
</table>
### 3.3 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 hard copy of the report.

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>Displays text, variable values, and/or column values after the specified set of rows (called a break specification). See &quot;DISPLAY&quot; (page 53) for details.</td>
</tr>
<tr>
<td>COMPUTE</td>
<td>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 &quot;COMPUTE&quot; (page 50) for details.</td>
</tr>
<tr>
<td>BREAK</td>
<td>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 &quot;BREAK&quot; (page 41) for details.</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Defines a variable and assigns a text value to it. When DISPLAY statements refer to the variable, ISQL prints the value. See &quot;DEFINE&quot; (page 52) for details.</td>
</tr>
<tr>
<td>COLUMN</td>
<td>Controls how ISQL displays a column's values (the FORMAT clause) and/or specifies alternative column-heading text (the HEADING clause). See &quot;COLUMN&quot; (page 45) for details.</td>
</tr>
<tr>
<td>TITLE</td>
<td>Specifies text and its positioning that ISQL displays before or after it processes a query. See &quot;TITLE&quot; (page 77) for details.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Removes settings made by the previous DISPLAY, COMPUTE, COLUMN, BREAK, DEFINE, or TITLE statements. See &quot;CLEAR&quot; (page 44) for details.</td>
</tr>
</tbody>
</table>

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 about outstanding customer orders. The query joins two tables, CUSTOMERS and ORDERS. The examples for the TABLE statement on "HOST or SH or !" (page 61) 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 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;
```

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterprises Inc.</td>
<td>Scottsdale</td>
<td>13</td>
<td>3000000</td>
</tr>
<tr>
<td>Aerospace Enterprises Inc.</td>
<td>Scottsdale</td>
<td>14</td>
<td>1500000</td>
</tr>
<tr>
<td>Chemical Construction Inc.</td>
<td>Joplin</td>
<td>11</td>
<td>3000000</td>
</tr>
<tr>
<td>Chemical Construction Inc.</td>
<td>Joplin</td>
<td>12</td>
<td>7500000</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>21</td>
<td>6000000</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>20</td>
<td>5000000</td>
</tr>
</tbody>
</table>

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

1. 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.

2. 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. "Summarizing Data with DISPLAY, COMPUTE, and BREAK Statements" (page 33) 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 "Adding Beginning and Concluding Titles with the TITLE Statement" (page 35).

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.
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 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;
```

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>13</td>
<td>3000000</td>
</tr>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>14</td>
<td>1500000</td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>11</td>
<td>3000000</td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>12</td>
<td>7500000</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>21</td>
<td>6000000</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>20</td>
<td>5000000</td>
</tr>
</tbody>
</table>

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 COLUMN (page 45).)

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 Customizing Format of Numeric Column Displays

```
ISQL> column order_value format "$99,999,999.99"
ISQL> column;
```

```
column CUSTOMER_CIT 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;

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>13</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>14</td>
<td>$1,500,000.00</td>
</tr>
<tr>
<td>Chemical Constructi</td>
<td>Joplin</td>
<td>11</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Chemical Constructi</td>
<td>Joplin</td>
<td>12</td>
<td>$7,500,000.00</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>21</td>
<td>$6,000,000.00</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>20</td>
<td>$5,000,000.00</td>
</tr>
</tbody>
</table>

Summarizing Data with 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_NAME 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 Specifying Column Breaks and Values with DISPLAY

```sql
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;
```

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>13</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Aerospace Enterprises</td>
<td>Scottsdale</td>
<td>14</td>
<td>$1,500,000.00</td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>11</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>12</td>
<td>$7,500,000.00</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>21</td>
<td>$6,000,000.00</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>20</td>
<td>$5,000,000.00</td>
</tr>
</tbody>
</table>

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

COMPUTE statements specify a c-treeSQL 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 further 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 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;

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterpris</td>
<td>Scottsdale</td>
<td>13</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Aerospace Enterpris</td>
<td>Scottsdale</td>
<td>14</td>
<td>$1,500,000.00</td>
</tr>
</tbody>
</table>
| 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

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 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

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>CUSTOMER_CITY</th>
<th>ORDER_ID</th>
<th>ORDER_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerospace Enterprise</td>
<td>Scottsdale</td>
<td>13</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Aerospace Enterprise</td>
<td>Scottsdale</td>
<td>14</td>
<td>$1,500,000.00</td>
</tr>
<tr>
<td>Summary of activity for Aerospace Enterprise</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of orders: 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total value of orders: $4,500,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>11</td>
<td>$3,000,000.00</td>
</tr>
<tr>
<td>Chemical Construction</td>
<td>Joplin</td>
<td>12</td>
<td>$7,500,000.00</td>
</tr>
<tr>
<td>Summary of activity for Chemical Construction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of orders: 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total value of orders: $10,500,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>21</td>
<td>$6,000,000.00</td>
</tr>
<tr>
<td>Luxury Cars Inc.</td>
<td>North Ridgeville</td>
<td>20</td>
<td>$5,000,000.00</td>
</tr>
<tr>
<td>Summary of activity for Luxury Cars Inc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of orders: 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total value of orders: $11,000,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tower Construction</td>
<td>Munising</td>
<td>8</td>
<td>$2,000,000.00</td>
</tr>
<tr>
<td>Tower Construction</td>
<td>Munising</td>
<td>10</td>
<td>$6,000,000.00</td>
</tr>
<tr>
<td>Tower Construction</td>
<td>Munising</td>
<td>9</td>
<td>$8,000,000.00</td>
</tr>
<tr>
<td>Summary of activity for Tower Construction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of orders: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total value of orders: $16,000,000.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

End of Orders Summary Report

23 records selected

ISQL>

3.4 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;
3.5 Transaction Support

A transaction is started with the execution of the first c-treeSQL 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 c-treeSQL 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 c-treeSQL 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 ;

A c-treeSQL statement starting immediately after a COMMIT WORK or ROLLBACK WORK statement starts a new transaction.

3.6 Work Flow Control

ISQL provides support for controlling the flow of execution in ISQL by means of the IF_EXISTS / IF_NOT_EXISTS keyword.

IF_EXISTS / IF_NOT_EXISTS imposes conditions on the execution of a single SQL statement or ISQL command or a block of SQL statements and ISQL commands. The SQL statement/statement block following an IF_EXISTS keyword and its condition is executed if the condition is satisfied, i.e., if the SELECT query returns at least one record.

The optional ELSE keyword introduces an alternate SQL statement or a block of SQL statements that is executed when the IF_EXISTS condition is not satisfied i.e., if the SELECT query returns no record. IF_NOT_EXISTS is just the reverse of IF_EXISTS, i.e., the SQL statement/statement block following an IF_NOT_EXISTS keyword and its condition is executed if the condition is not satisfied, i.e., if the SELECT query returns no record.
IF_EXISTS / IF_NOT_EXISTS controls can be nested. There is no limit to the number of nested levels. If the select query in the condition returns an error due to any reason, the error is returned and none of the conditional blocks is executed.
3.7 **ISQL Reference**

This section provides reference material for statements specific to ISQL.

This section does not include descriptions of standard c-treeSQL statements or statements specific to embedded c-treeSQL. For details on the syntax and semantics of those other c-treeSQL statements, refer to the *c-treeSQL Reference Manual*. 

@ (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 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 the file cmdfile, enter:

ISQL> @cmdfile
## 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:

<table>
<thead>
<tr>
<th><code>column_name</code></th>
<th>Causes a break when the value of the column specified by <code>column_name</code> changes.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ROW</code></td>
<td>Causes a break on every row selected by a <code>SELECT</code> statement.</td>
</tr>
<tr>
<td><code>PAGE</code></td>
<td>Causes a break at the end of each page. The end of a page is specified in the SET PAGESIZE statement. See &quot;SET (page 68)&quot; for details on the SET statement.</td>
</tr>
<tr>
<td><code>REPORT</code></td>
<td>Causes a break at the end of a report or query.</td>
</tr>
</tbody>
</table>

`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>
**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 - 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 customer_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>
```
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.

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Number of 9s specifies width. If the column value is too large to display in the specified format, ISQL displays # characters in place of the value.</td>
</tr>
<tr>
<td>0</td>
<td>Display leading zeroes.</td>
</tr>
</tbody>
</table>

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

Numeric Format Strings for the COLUMN Statement

<table>
<thead>
<tr>
<th>Character</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>99999</td>
<td>Number of 9s specifies width. If the column value is too large to display in the specified format, ISQL displays # characters in place of the value.</td>
</tr>
<tr>
<td>0</td>
<td>09999</td>
<td>Display leading zeroes.</td>
</tr>
</tbody>
</table>
### ISQL Statements

**Character** | **Example** | **Description**
---|---|---
$ | $9999 | Prefix the display with ‘$’.  
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 ‘>’.  

**Date-Time Format Strings for the COLUMN Statement**

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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).  
| 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).  

---
<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHHH24</td>
<td>The hour value as a 2-digit number (in the range 00 to 23).</td>
</tr>
<tr>
<td>MI</td>
<td>The minute value as a 2-digit number (in the range 00 to 59).</td>
</tr>
<tr>
<td>SS</td>
<td>The seconds value as a 2-digit number (in the range 00 to 59).</td>
</tr>
<tr>
<td>SSSSS</td>
<td>The seconds from midnight as a 5-digit number (in the range 00000 to 86399).</td>
</tr>
<tr>
<td>MLS</td>
<td>The milliseconds value as a 3-digit number (in the range 000 to 999).</td>
</tr>
</tbody>
</table>

**Examples**

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

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL ?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>customer_id</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>steel_type</td>
<td></td>
<td>CHAR</td>
<td>20</td>
</tr>
<tr>
<td>order_info</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>order_weight</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>order_value</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>order_state</td>
<td></td>
<td>CHAR</td>
<td>20</td>
</tr>
</tbody>
</table>

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
```

---

Illustrate some options with numeric format strings.

```
ISQL> select order_value from orders where order_value < 1000000;
ORDER_VALUE
----------

```

---

No column formatting:

```
ISQL> select order_value from orders where order_value < 1000000;
ORDER_VALUE
----------

```
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
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 in "BREAK" (page 41) 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;
<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>ORDER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sports Cars Inc.</td>
<td></td>
</tr>
<tr>
<td>Sports Cars Inc.</td>
<td></td>
</tr>
<tr>
<td>Mighty Bulldozer Inc.</td>
<td></td>
</tr>
<tr>
<td>Mighty Bulldozer Inc.</td>
<td></td>
</tr>
</tbody>
</table>

Number of orders placed by Sports Cars Inc. = 2
Number of orders placed by Mighty Bulldozer Inc. = 2
**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" ;
```
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:

<table>
<thead>
<tr>
<th>COL column_number</th>
<th>Directly specifies the column position of the display value as an integer(1 specifies column 1, 2 specifies column 2, and so on.).</th>
</tr>
</thead>
<tbody>
<tr>
<td>@column_name</td>
<td>Names a column in the select list of the SQL query. ISQL aligns the display value with the specified column.</td>
</tr>
</tbody>
</table>

  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 in "BREAK" (page 41) 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;
```

<table>
<thead>
<tr>
<th>CUSTOMER_NAME</th>
<th>ORDER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sports Cars Inc.</td>
<td>1</td>
</tr>
<tr>
<td>Sports Cars Inc.</td>
<td>2</td>
</tr>
<tr>
<td>Sports Cars Inc.</td>
<td></td>
</tr>
<tr>
<td>Number of orders placed by Sports Cars Inc.</td>
<td>2</td>
</tr>
<tr>
<td>Mighty Bulldozer Inc.</td>
<td>3</td>
</tr>
<tr>
<td>Mighty Bulldozer Inc.</td>
<td>4</td>
</tr>
<tr>
<td>Mighty Bulldozer Inc.</td>
<td></td>
</tr>
<tr>
<td>Number of orders placed by Mighty Bulldozer Inc.</td>
<td>2</td>
</tr>
<tr>
<td>Ship Shapers Inc.</td>
<td>5</td>
</tr>
<tr>
<td>Ship Shapers Inc.</td>
<td>6</td>
</tr>
<tr>
<td>Ship Shapers Inc.</td>
<td>7</td>
</tr>
<tr>
<td>Ship Shapers Inc.</td>
<td></td>
</tr>
<tr>
<td>Number of orders placed by Ship Shapers Inc.</td>
<td>3</td>
</tr>
<tr>
<td>Tower Construction Inc.</td>
<td>8</td>
</tr>
<tr>
<td>Tower Construction Inc.</td>
<td>9</td>
</tr>
<tr>
<td>Tower Construction Inc.</td>
<td>10</td>
</tr>
<tr>
<td>Tower Construction Inc.</td>
<td></td>
</tr>
<tr>
<td>Number of orders placed by Tower Construction Inc.</td>
<td>3</td>
</tr>
</tbody>
</table>

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
```
EDIT

Syntax
ED[IT] [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 Windows. 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 Windows, set the environment variable in the initialization file `DHSQL.INI` in the `%WINDIR%` 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.
```

```
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.
GET

Syntax
G(GET) 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,
ISQL Statements

```
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
```
HELP

Syntax
HELP [LP] {COMMANDS|CLAUSES} ;
HELP [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;
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'
```
HOST or SH or !

Syntax

{ HOST | SH | ! } [host_command];

Description

The HOST (or SH or !) statement executes a host operating system command without terminating the current ISQL session.

Arguments

host_command

The operating system command to execute. If host_command is not specified, ISQL spawns a sub-shell 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:

```
-- Check the syntax for the UNIX 'more' command:
ISQL> host more
Usage: more [-dfln] [+linenum | +/pattern] name1 name2 ...
```

```
-- 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 ) )
;
```

```
-- Spawn a subshell process to issue multiple OS commands:
ISQL> sh
```

**IF [ NOT ] EXISTS**

**Syntax**

```
{IF [NOT] EXISTS} | {IF_EXISTS} | {IF_NOT_EXISTS} (<query>)
{BEGIN
    <list of statements>
END} | { <statement> }

[ELSE
    {BEGIN
        <list of statements>
    END} | { <statement> } ]
```

Conditional expressions can be nested. The following example checks for the existence of a table and creates it if it does not exist. The nested condition updates an existing table if it is required.

**Example**

```sql
IF NOT EXISTS (SELECT * FROM systables where tbl = 'custmast')
BEGIN
    CREATE TABLE custmast (name CHAR(10), custid INTEGER IDENTITY (1,1), balance MONEY, modtime TIMESTAMP);
END
ELSE
BEGIN
    IF NOT EXISTS (SELECT * FROM syscolumns where tbl='custmast' AND col='modtime')
    BEGIN
        ALTER TABLE custmast ADD (modtime TIMESTAMP);
    END
    ELSE
    BEGIN
        SELECT * FROM custmast;
    END
END
COMMIT WORK;
```

The following keywords have been added to the ISQL lexicon to support conditional expressions:

- `IF`
- `IF_EXISTS`
- `IF_NOT_EXISTS`
- `BEGIN`
- `ELSE`
- `END`

Statements that use these keywords as identifiers should enclose them in double quotes.
INDEXES

Syntax
I[INDEXES] [ tablename ] ;

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

With the tablename argument, the INDEX statement displays a brief description of the indexes defined on the specified table.

The index list is sorted by table name and index name in that order.

Example
The following example uses the index list command to display lists of indexes defined on user tables.

```
ISQL> INDEXES
INDEXNAME  TABLENAME  ID  OWNER  TYPE  ORDER  COMPRESSION  METHOD
---------  ---------  ---  -----  ----  -----  -----------  ------
sys_001_000000041 dept  33  admin  U    A      N            B
dno_idx01    emp  35  admin  D    A      N            B
sys_001_000000042 emp  34  admin  U    A      N            B

ISQL> INDEXES emp ;
INDEXNAME  TABLENAME  ID  OWNER  TYPE  ORDER  COMPRESSION  METHOD
---------  ---------  ---  -----  ----  -----  -----------  ------
deptno_idx01 emp  36  admin  D    A      N            B
name_idx01    emp  34  admin  D    A      N            B
name_idx02    emp  35  admin  D    A      N            B
sys_001_000000041 emp  33  admin  U    A      N            B
```
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>
```
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.
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 RUN 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>
SAVE

Syntax
SAVE 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 ADMIN         51 May  1 18:21 test.sql
ISQL> ! more test.sql
select customer_name, customer_city from customers
ISQL>
**SET**

**Syntax**

```sql
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 c-treeSQL session.

**Arguments**

**HISTORY**

Sets the number of statements that ISQL will store in the history buffer. The default is 1 statement and the maximum is 250 statements.

**PAGESIZE number_lines**

Sets the number of lines per page. The default is 24 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 80 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 1.

**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 c-treeSQL statements, so that only results are displayed. SET ECHO ON is the default.

**PAUSE ON | OFF**

All Rights Reserved
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 c-treeSQL optimizer uses to calculate the least-costly query strategy for a particular c-treeSQL 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 OFF is the default. SET AUTOCOMMIT OFF requires that you end transactions explicitly with a COMMIT or ROLLBACK WORK statement.

**Note:** Be careful using the automatic commit logic when working with cursors. When automatic commit is enabled, the logic will execute a commit operation after each database access, which will close any open cursor. If you receive an error -20039 (Open for non-select statement) then most likely your cursor has been closed.

**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. The default is 3. See the SET TRANSACTION ISOLATION LEVEL statement in the *c-treeSQL 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 *c-treeSQL 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 c-treeSQL writes output is controlled by the SPOOL filename ON statement.

**Examples**

```sql
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
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 "SET" (page 68), "SPOOL" (page 72), and "EDIT" (page 55) 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

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>CONNECTION NAME</th>
<th>IS DEFAULT?</th>
<th>IS CURRENT?</th>
</tr>
</thead>
<tbody>
<tr>
<td>salesdb</td>
<td>conn_1</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
**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 ;
```
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 argument:
ISQL> START start_ex.sql Ship
CUSTOMER_NAME
------------
Ship Shapers Inc.
1 record selected
ISQL> -- ISQL puts the script statement, after argument substitution, 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%'

### 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 c-treeSQL database administrator (the user ADMIN, by default), you need to qualify the system table name with the administrator user name, as in the following example:

```
ISQL> table ADMIN.systables
```

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL ?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td>NOT NULL</td>
<td>INT</td>
</tr>
<tr>
<td>tbl</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>creator</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>owner</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>tbltype</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>tblpctfree</td>
<td></td>
<td>NOT NULL</td>
<td>INT</td>
</tr>
<tr>
<td>segid</td>
<td></td>
<td>NOT NULL</td>
<td>INT</td>
</tr>
<tr>
<td>has_pcnstrs</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>has_fcnstrs</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>has_ccnstrs</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>has_ucnstrs</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>tbl_status</td>
<td></td>
<td>NOT NULL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>rssid</td>
<td></td>
<td>NOT NULL</td>
<td>INT</td>
</tr>
</tbody>
</table>

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

```
ISQL> table customers
```

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL ?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>customer_name</td>
<td></td>
<td>CHAR</td>
<td>50</td>
</tr>
<tr>
<td>customer_street</td>
<td></td>
<td>CHAR</td>
<td>100</td>
</tr>
<tr>
<td>customer_city</td>
<td></td>
<td>CHAR</td>
<td>50</td>
</tr>
</tbody>
</table>
ISQL> table orders
<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>customer_id</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>steel_type</td>
<td></td>
<td>CHAR</td>
<td>20</td>
</tr>
<tr>
<td>order_info</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
<tr>
<td>order_weight</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>order_value</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>order_state</td>
<td></td>
<td>CHAR</td>
<td>20</td>
</tr>
</tbody>
</table>

ISQL> table lots
<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>lot_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>order_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>lot_units</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>lot_info</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
</tbody>
</table>

ISQL> table lot_staging
<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>lot_id</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>lot_location</td>
<td></td>
<td>CHAR</td>
<td>20</td>
</tr>
<tr>
<td>start_date</td>
<td></td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>end_date</td>
<td></td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>issues</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
</tbody>
</table>

ISQL> table quality
<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>lot_id</td>
<td>NOT NULL</td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>purity</td>
<td></td>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>p_deviation</td>
<td></td>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>strength</td>
<td></td>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>s_deviation</td>
<td></td>
<td>DOUBLE</td>
<td>8</td>
</tr>
<tr>
<td>comments</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
</tbody>
</table>

ISQL> table samples
<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NULL?</th>
<th>TYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>lot_id</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>samples</td>
<td></td>
<td>INT</td>
<td>4</td>
</tr>
<tr>
<td>comments</td>
<td></td>
<td>CHAR</td>
<td>200</td>
</tr>
</tbody>
</table>
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 n 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
```
4. Data Load Utility: dbload

This chapter describes the c-treeSQL database load utility, dbload. This utility loads records from an input data file into tables of a database. The format of the data file is specified by a record description given in an input commands file to dbload.

The commands files of both dbload and dbdump use DEFINE RECORD statements with similar syntax to specify the format of loaded or exported data records. The commands files specify the data file, the format of data records, and the destination (or source) database columns and tables for the data.

The dbload utility allows loading of variable- or fixed-length records, and lets the load operation specify the set of fields and records to be stored from an input file. Data files can use multiple-character record delimiters. dbload also allows control of other characteristics, such as error handling and logging, in its command line. dbload generates a badfile that contains records from the input file that failed to load in the database.

The following figure shows the dbload execution process.
4.1 Prerequisites for dbload

Before running dbload, you need:

- A valid, readable commands file
- INSERT privileges on the tables named in the commands file

4.2 dbload Command Line Syntax

The dbload command does not directly specify an input file, but instead names a commands file that in turn specifies data input files. The dbload command accepts the commands file name, the database name, and a list of command options.

Syntax

dload -f commands_file [ options ] database_name

- `-f commands_file` - Specifies the file containing dbload commands.
- `options` - One or more of the options as described below
- `database_name` - Name of the database.

Options

- `-u user_name` - The user name c-treeSQL uses to connect 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 c-treeSQL uses to connect 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.)
- `-z maximum multiple inserts` - The maximum number of records to be inserted at one time in each bulk insert (used to improve performance)
- `-l logfile` - Specifies the file into which the error logging is done. stderr is the default.

dbload also writes statistics to the file:

- Number of records read
- Number of records skipped
- Number of records loaded
- Number of records rejected

- `-b badfile` - The file into which the bad rows that were not loaded, are written. By default badfile is put in the current directory.
- `-c commit_frequency` - Store the specified number of records before committing the transaction. The default frequency is 100 records.
- `-e maxerrs` - The maximum number of tolerable errors. The default number is 50 errors.
- `-s skipcount` - Skip the specified number of rows in the first data file. If multiple files are specified, the rows are skipped only in the first file. The default number is zero rows.
- `-m maxrows` - Stop storing rows at the specified number.
Data Load Utility: dbload

- Parse the commands file and display errors, if any, without doing the database load. If the parsing is successful, the following message displays on *stdout*: "No errors in the commands file."

**Unicode**

The *dbdump* Data Unload and *dbload* Data Load utilities properly dump and load data out of a Unicode-enabled c-treeACE SQL Server. The commands file must be in ASCII format. The output file generated by *dbdump* is in Unicode format with an optional Byte Order Mark (BOM) indicating the Unicode encoding form (using the `-B` command-line switch).

The input file for *dbload* needs to be in Unicode (native "wchar" encoding form) with an optional BOM, in which case the utilities check for the proper format.

### 4.3 Data File Formats

Data files must be in one of the following record formats:

- Variable length records
- Fixed length records

For both these types of records an optional field delimiter and an optional record delimiter can be specified. The field delimiter, when specified, should be a single character. By default, comma is the field delimiter. The record delimiter can be specified in the commands file and it can be more than one character. By default, the newline character, \n, is the record delimiter.

**Variable Length Records**

For variable length records, the fields in the data file can be of varying length. Unless the keyword FIXED is used in the commands file, it is assumed that the *dbload* record processing will be for variable length records.

**Fixed Length Records**

For fixed length records, the fields in the data file must be of fixed length. The length of the record must be the same for all records and is specified in the commands file. In case of fixed length records, the field and record delimiters are ignored. That is, the POSITION specification must be such that the delimiters are ignored. For more information on the commands file refer to "The Commands File" (page 81).

The data files that contain fixed length records can either be ASCII or binary files.

### 4.4 The Commands File

The commands file specifies instructions for *dbload* to load the records into the table specified. Thus the commands file defines what *dbload* will be performing for a particular loading process.

There is no file naming convention for the commands file. For example, the commands file name to load the ORDERS table could be *ORDERS_CMD*. 

---

All Rights Reserved 81 www.faircom.com
The commands file must contain the following parts:

- The DEFINE RECORD statement
- The FOR EACH statement

The syntax definition for the commands file is as shown:

```
dbload_commands:
    define_record_statement
    for_each_statement
```

The following is sample commands file showing load instructions.

```
DEFINE RECORD ord_rec AS
    ( ord_no, item_name, date, item_qty ) FIELD DELIMITER ' ' ;

FOR EACH RECORD ord_rec FROM ord_in
    INSERT INTO ADMIN.orders
        (order_no, product, order_date, qty)
    VALUES (ord_no, item_name, date, item_qty) ;

NEXT RECORD
```

The above commands specification instructs `dbload` to load records into the orders table. The fields in the data file, `ORD_IN`, appear in the order listed in the DEFINE RECORD statement.

### The DEFINE RECORD Statement

The DEFINE RECORD statement is used to define the record that is to be loaded into the database. It describes the data found in the data file. The following are the definitions that are made known by the DEFINE RECORD statement:

- Names the record to be loaded
- Names the fields of the record to be loaded as found in the data file
- Specifies whether the records in the data file are variable length records or fixed length records
- If fixed length records, specifies the position and data type of the field

The following is the syntax definition of the DEFINE RECORD statement:

```
DEFINE RECORD record_name
    [ OF FIXED LENGTH record_length
    AS { 
    field_name position_specification type_specification, ...
    } 

    AS { 
    field_name, ...
    }

    [ FIELD DELIMITER delimiter_char ]
    [ RECORD DELIMITER delimiter_string ]

position_specification::
    POSITION ( start_position : end_position )

type_specification::
    CHAR
```
The following are the variable descriptions of the DEFINE RECORD syntax:

- **record_name** is the name used to refer to the records found in the data file.
- **record_length** is the length of the fixed length record. This length should include the length of field or record delimiters, if any.
- **field_name** is the name used to refer to a field in the data file.
- **delimiter_char** is the field delimiter and is a single character. It must be specified as a literal.
- **delimiter_string** is the record delimiter and can be a single character or a string. It must be specified as a literal.
- **start_position** is the position where the field starts. It must be an unsigned integer.
- **end_position** is the position where the field ends. It must be an unsigned integer.

The first position of each record is 1 and not 0.

If date and time types are to be inserted they can be specified as characters in the data file. If it is a fixed length record then the type specification can be CHAR.

The following is an example of the DEFINE RECORD statement for fixed length records:

```sql
DEFINE RECORD rec_one OF FIXED LENGTH 20
AS (
   fld1 POSITION (1:4) SHORT,
   fld2 POSITION (5:15) CHAR,
   fld3 POSITION (16:20) CHAR
) ;
```

### The FOR EACH Statement

The FOR EACH statement scans for each valid record in the data file and inserts the record into the database. The syntax for the FOR EACH statement is shown below:

```sql
FOR EACH RECORD record_name FROM data_file_name, ...
    INSERT INTO owner_name.target_table [ {field_name, ...} ]
    VALUES (value, ...); NEXT RECORD
```

The following are the variable descriptions of the FOR EACH statement:

- **record_name** is the record name that is specified in the DEFINE RECORD statement.
- **data_file_name** is the name of the input data file name.
- **owner_name.target_table** is the target table name identified along with the owner name of the table. The **target_table** must already exist in the database and must have appropriate permissions for inserting the records.
• **field_name** is the name of the field or column in the table.
• **value** is the value that must be inserted into the table.

The **target_table** can also be a synonym on another table with the INSERT access. The list of values that are to be inserted must follow the VALUES keyword. The values that can be inserted are:

• Name of the field in the input data file
• A constant (both numeric as well as character)
• NULL

The values specified in the VALUES list must correspond one to one with that in the target table list. The list can be in any order compared to the list specified in the DEFINE RECORD statement. The following example shows the list interchanged with respect to the list in the DEFINE RECORD statement.

```sql
DEFINE RECORD dept_rec AS
   ( dept_no, dept_name, location ) FIELD DELIMITER ' ' ;

FOR EACH RECORD dept_rec FROM dept_in
   INSERT INTO ADMIN.department (loc, no, name)
   VALUES (location, dept_no, dept_name) ;

NEXT RECORD
```

Here the items **no**, **name**, and **loc** are interchanged in both the table list and the values list when compared with the DEFINE RECORD list.

The keyword NEXT RECORD must be specified after the FOR EACH statement so that the insert loop is terminated.

### 4.5 Examples

This section gives different types of examples for **dbload**, both for variable length records as well as fixed length records. The data files can either be ASCII or binary files. If they are binary files they must be in the fixed length record format.

The following example is the commands file to load records into the DEPT table. The input data file name is **deptrecs_in** which is an ASCII file in the variable length record format.

**Example dbload commands files**

```sql
DEFINE RECORD dept_rec AS
   ( dept_no, dept_name, location ) FIELD DELIMITER ' ' ;

FOR EACH RECORD dept_rec FROM deptrecs_in
   INSERT INTO ADMIN.dept (no, name, loc)
   VALUES (dept_no, dept_name, location) ;

NEXT RECORD
```
The following is the commands file to load records into the CUSTOMER table. The input data file is `cust_in` which is a binary file in the fixed length record format.

```sql
DEFINE RECORD cust_rec OF FIXED LENGTH 36
AS (  
cust_no POSITION (1:4) LONG,  
cust_name POSITION (5:15) CHAR,  
cust_street POSITION (16:28) CHAR,  
cust_city POSITION (29:34) CHAR,  
cust_state POSITION (35:36) CHAR  
) ;
FOR EACH RECORD cust_rec FROM cust_in  
    INSERT INTO ADMIN.customer (no, name, city, street, state)  
    VALUES (cust_no, cust_name, cust_city, cust_street, 'CA') ;
NEXT RECORD
```

The following is the commands file to load records into the ORDERS table. The input data file is `orders_in` which is a binary file in the fixed length record format.

```sql
DEFINE RECORD orders_rec OF FIXED LENGTH 30
AS (  
order_no POSITION (1:4) LONG,  
order_date POSITION (6:16) CHAR,  
product POSITION (18:25) CHAR,  
qty POSITION (27:30) LONG  
) ;
FOR EACH RECORD orders_rec FROM orders_in  
    INSERT INTO ADMIN.orders (no, date, prod, units)  
    VALUES (order_no, order_date, product, qty) ;
NEXT RECORD
```

### 4.6 dbload Errors

This section discusses the different types of errors that can occur during the execution of `dbload`. There are three types of errors that can occur during the `dbload` execution process:

- Commands file errors
- `dbload` errors
- c-treeACE SQL database errors

The invalid records that are encountered during the processing of records from the data files are flagged as bad records and are written to the `badfile` that is specified in the `dbload` command option. By default, the bad records are written to the file, `badfile`, in the current directory. Any error in the input data file is messaged in the log file (if specified in the command line option) along with the statistics. The following sections discuss the compilation errors and fatal errors that could occur during the `dbload` process execution.
Compilation Errors

The compilation error messages are as follows:

Record name redefined.
The record name in the DEFINE RECORD statement was already defined. The record name must be unique. dbload creates a new definition using the same name.

Error in record definition.

Too many fields in record definition.
The number of fields used in the record definition is more than the maximum allowed. Currently, the maximum number allowed is $TPE\_MAX\_FIELDS$ in the header file $sql\_lib.h$.

Position not specified for fixed length record.

Position for SHORT not specified correctly.
The size of the field (start position to end position) must be equal to the size of SHORT.

Position for LONG not specified correctly.
The size of the field (start position to end position) must be equal to the size of LONG.

Position for FLOAT not specified correctly.
The size of the field (start position to end position) must be equal to the size of FLOAT.

Position for DOUBLE not specified correctly.
The size of the field (start position to end position) must be equal to the size of DOUBLE.

Field delimiter must be a single character.

Invalid record delimiter.

Record not defined.
The FOR EACH statement is used with a record name that is not defined.

Mismatch in value list.
The number of values specified in the VALUES list does not match with that specified in the DEFINE RECORD list.

Too many data files specified.
Currently, the maximum number of data files that can be specified in a FOR EACH statement is 10.

Column not found in record definition.

Fatal Errors

The following are a list of nonrecoverable errors.

- No memory
- Table not found
- No columns in the table
• Column not found
• Too many fields
• More than the maximum number of fields allowed, is specified in the table list of the FOR EACH statement.
• Cannot open <bad file name>
• Cannot open <data file name>
• Cannot open log file <log file name>

The dbload execution process can also stop if the number of tolerable errors specified (-e option) on the command option is exceeded. By default the number of tolerable errors is 50.
5. Data  Unload Utility: dbdump

This chapter describes the c-treeSQL database dump utility, dbdump.

dbdump writes the data in a database to a file. The format of the exported data is specified by the record description given in an input command file to dbdump.

Both dbload and dbdump commands files use DEFINE RECORD statements with similar syntax to specify the format of loaded or exported data records. The commands file specifies the data file, the format of data records, and the destination (or source) database columns and tables for the data.

Prerequisites for dbdump
Before running dbdump, you need:

- A valid, readable commands file
- SELECT privileges on the tables named in the commands file

5.1 dbdump Command-Line Syntax

The dbdump command accepts the commands file name, the database name and a command option.

Syntax

```
```

Options

- `-f commands_file` - Specifies the file containing dbdump commands.
- `-u user_name` - The user name c-treeSQL uses to connect 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 c-treeSQL uses to connect 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.)
- `-n` - Parse the commands file and display errors, if any, without exporting data. If the parsing is successful, the following message displays on stdout:

  *No errors in the commands file.*

- `-z` (V10.3 and later) - The number of records to fetch per call is controlled by the -z switch. To be consistent with the switch of dbload, it defaults to 1.
- The `-l` switch controls the progress output frequency in terms of record reads (the output is generated whenever the number of record reads since the last output becomes greater than or equal to the specified value).

- The `dbdump -p` command-line switch activates the new query passthru mechanism, which does not perform any major change in the query. When this switch is in use, `dbdump` interprets the query in the command file, converts it to lowercase, and wraps everything that is considered an identifier with double-quotes. This can cause a syntax error if the command file contains functions. For example, consider the following statement:

```sql
SELECT RTRIM(cm_custnumb),...
```

The default behavior of `dbdump` results in the following statement, which would cause a syntax error:

```sql
select "rtrim" ( "cm_custnumb" ) ,...
```

Using the `dbdump -p` switch results in this statement, which does not cause an error:

```sql
select rtrim ( cm_custnumb ) ,...
```

- **database_name** - Name of the database.

- **-B** - Use BOM (Unicode Byte Order Mark) in output file.

### Automatic Record Definition

Customers have a need to export all data in a CSV format. `dbdump` is great for this purpose. (In general, `dbdump` primary usage is with SELECT * FROM ...) However, with large tables, there is a lot of setup work to create the DEFINE RECORD information. For example, for a table with more than 100 columns, they have to manually describe the complete output structure for all fields.

In V11.5 and later we have introduced a new syntax in the DBDUMP script to automatically generate the record definition. The syntax definition for the commands file is as shown:

```
dbdump_commands:
define_record_statement
for_record_statement

where define_record_statement is either the following (existing syntax):

DEFINE RECORD record_name
[ OF FIXED LENGTH record_length
AS ( field_name position_specification
type_specification,
...
)

] [ FIELD DELIMITER delimiter_char ]
[ RECORD DELIMITER delimiter_string ]
```

or the following (new added syntax):

```
AUTODEFNE RECORD record_name
[ FIELD DELIMITER delimiter_char ]
[ RECORD DELIMITER delimiter_string ]
[FIELDS PER LINE number ];
```
Unicode

The dbdump Data Unload and dbload Data Load utilities properly dump and load data out of a Unicode-enabled c-treeACE SQL Server. The commands file must be in ASCII format. The output file generated by dbdump is in Unicode format with an optional Byte Order Mark (BOM) indicating the Unicode encoding form (using the -B command-line switch).

The input file for dbload needs to be in Unicode (native "wchar" encoding form) with an optional BOM, in which case the utilities check for the proper format.

5.2 Data File Formats

The output data file can be defined to be having one of the following record formats:

- Variable length records
- Fixed length records

For both these types of records an optional field delimiter and an optional record delimiter can be specified. The field delimiter, when specified, should be a single character. By default, comma is the field delimiter. The record delimiter can be specified in the commands file and it can be more than one character. By default, the newline character, \n, is the record delimiter.

Variable Length Records

For variable length records, the fields in the data file can be of varying length. Unless the keyword FIXED is used in the commands file, it is assumed that the dbload record processing will be for variable length records.

Fixed Length Records

For fixed length records, the fields in the data file must be of fixed length. The length of the record must be the same for all records and is specified in the commands file. In case of fixed length records, the field and record delimiters are ignored. That is, the POSITION specification must be such that the delimiters are ignored. For more information on the commands file refer to "The Commands File" (page 81).

The data files that contain fixed length records can either be ASCII or binary files.

5.3 The Commands File

The commands file specifies:

- Record format for the output file
- Query which is to be used for exporting data

There is no file naming convention for the commands file. For example, the commands file name to load the ORDERS table could be orders.cmd.

The commands file must contain the following parts:
The DEFINE RECORD statement

The FOR RECORD statement

The syntax definition for the commands file is as shown:

dbdump_commands:
  define_record_statement
  for_record_statement

The following is sample commands file showing dump instructions.

DEFINE RECORD ord_rec AS
  ( ord_no, item_name, date, item_qty ) FIELD DELIMITER ' ' ;

FOR RECORD ord_rec dump into ord_dat
USING SELECT order_no, product, order_date, qty
FROM items;

The DEFINE RECORD Statement

The DEFINE RECORD statement is used to define the record of the output file. The following are the definitions that are made known by the DEFINE RECORD statement:

- Names the record of the output file
- Names the fields of the record
- Specifies whether the records in the data file are variable length records or fixed length records
- If fixed length records, specifies the position and data type of the field

The following is the syntax definition of the DEFINE RECORD statement:

DEFINE RECORD record_name
  [ OF FIXED LENGTH record_length
  AS { 
    field_name position_specification type_specification,
    ...
  } ]
  [ FIELD DELIMITER delimiter_char ]
  [ RECORD DELIMITER delimiter_string ] ;

position_specification::
  POSITION ( start_position : end_position )

type_specification::
  | CHAR
  | SHORT
  | LONG
  | FLOAT
  | DOUBLE
The following are the variable descriptions of the DEFINE RECORD syntax:

- **record_name** is the name used to refer to the records found in the data file.
- **record_length** is the length of the fixed length record. This length should include the length of field or record delimiters, if any.
- **field_name** is the name used to refer to a field in the data file.
- **delimiter_char** is the field delimiter and is a single character. **delimiter_char** must be specified as a literal.
- **delimiter_string** is the record delimiter and can be a single character or a string. It must be specified as a literal.
- **start_position** is the position where the field starts. It must be an unsigned integer.
- **end_position** is the position where the field ends. It must be an unsigned integer.

The first position of each record is 1 and not 0.

If **DATE**, **TIME**, and **TIMESTAMP** types are to be dumped they can be specified as characters in the commands file. If it is a fixed length record then the type specification can be **CHAR**.

The following is an example of the DEFINE RECORD statement for fixed length records:

```
DEFINE RECORD rec_one OF FIXED LENGTH 20
AS {
    fld1  POSITION (1:4)   SHORT,
    fld2  POSITION (5:15)  CHAR,
    fld3  POSITION (16:20) CHAR
} ;
```

See also:
- **Automatic Record Definition** in the topic titled **dbdump Command-Line Syntax** (page 88).

### The FOR RECORD Statement

The FOR RECORD statement writes each valid record into the data file after selecting the record from the database. The syntax for the FOR RECORD statement is shown below:

```
FOR  RECORD record_name DUMP INTO data_file_name
    USING select_statement ;
```

The following are the variable descriptions of the FOR RECORD statement:

- **record_name** specifies the same name used in the associated DEFINE RECORD statement.
- **data_file_name** is the name of the output data file name.
- **select_statement** is any valid **SELECT** statement.
5.4 Examples

This section gives different types of examples for dbdump, both for variable length records as well as fixed length records. The data files can either be ASCII or binary files. If they are binary files they must be in the fixed length record format.

The following shows the commands file to write records from the DEPT table. The output data file name is deptrecs_out which is an ASCII file in the variable length record format.

```
DEFINE RECORD dept_rec AS
    ( no, name, loc ) FIELD DELIMITER ' ' ;

FOR RECORD dept_rec DUMP INTO deptrecs_out
    USING SELECT dept_no , dept_name , location
FROM ADMIN.dept ;
```

The following shows the commands file to write records from the CUSTOMER table. The output data file is cust_out which is a binary file in the fixed length record format.

```
DEFINE RECORD cust_rec OF FIXED LENGTH 37 AS
    ( no POSITION (1:4) LONG,
      name POSITION (5:15) CHAR,
      street POSITION (16:28) CHAR,
      city POSITION (29:34) CHAR,
      state POSITION (35:36) CHAR ) ;

FOR RECORD cust_rec DUMP INTO cust_out
    USING SELECT cust_no, cust_name, cust_city, cust_street, cust_state
FROM ADMIN.customer ;
```

The following shows the commands file to dump records from the ORDERS table. The output data file is orders_out which is a binary file in the fixed length record format.

```
DEFINE RECORD orders_rec OF FIXED LENGTH 31 AS
    ( no POSITION (1:4) LONG,
      date POSITION (6:16) CHAR,
      prod POSITION (18:25) CHAR,
      units POSITION (27:30) LONG ) ;

FOR RECORD orders_rec DUMP INTO orders_out
    USING SELECT order_no, order_date, product, quantity
FROM ADMIN.orders ;
```

The c-treeACE SQL utility, **dbschema**, recreates specified database elements and data.

**Syntax**

```
dbschema [ -h ] [ -d ] [-u user_name ] [-a password ] [ -o outfile ]
   [-p { user_name.}procedure_name [ , ... ] ]
   [-t { user_name.}table_name [ , ... ] ]
   [-T { user_name.}trigger_name [ , ... ] ]
   [ database_name ]
```

**Description**

Generates c-treeSQL statements to recreate the specified database elements and data. If the **dbschema** statement omits all arguments, it displays definitions for all elements (tables, views, indexes, procedures, and triggers) for the default database on the screen.

**Options**

- **-h** - Displays brief online help of **dbschema** syntax and options.
- **-d** - In conjunction with the **-t** option, specifies that **dbschema** generates c-treeSQL INSERT statements for data in the tables, in addition to CREATE statements. The output of the **dbschema** command invoked with the **-d** option can be directed to a command file and executed in interactive c-treeSQL to duplicate and load table definitions.
- **-u user_name** - The user name c-treeSQL uses to connect to the database. c-treeSQL 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 c-treeSQL uses to connect to the database. c-treeSQL 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.)
- **-o outfile** - Redirects the output to the specified file. The default is standard output.
- **-t { user_name.}table_name [ , ... ]** - A comma-separated list of tables and views for which definitions should be generated. Specify a list of specific tables, or use the % to generate definitions for all tables.

**Note:** With the **-t** option, the % character is not a true wildcard character. It substitutes for the entire **table_name** argument and cannot be used for pattern matching within a character string. This differs from the behavior of the % in the **-p** and **-T** options.

By default, **dbschema** generates definitions for tables owned by the current user. Use the optional **user_name** qualifier to specify a table owned by a different user.

- **-p { user_name.}procedure_name [ , ... ]** - A comma-separated list of stored procedures for which definitions should be generated. The table names in the list can include the ‘%’ and underscore, ‘_’, characters, which provide pattern-matching semantics:
The ‘%’ matches zero or more characters in the procedure name
• The underscore ‘_’ matches a single character in the procedure name

By default, `dbschema` generates definitions for procedures owned by the current user. Use the optional `user_name` qualifier to specify a procedure owned by a different user.

• `-T [ user_name.][trigger_name [ , ... ] - A comma-separated list of triggers for which definitions should be generated. The table names in the list can include the ‘%’ and underscore ‘_’ characters, which provide pattern-matching semantics:
  • The ‘%’ matches zero or more characters in the trigger name
  • The underscore ‘_’ character matches a single character in the trigger name

By default, `dbschema` generates definitions for triggers owned by the current user. Use the optional `user_name` qualifier to specify a trigger owned by a different user.

• `database_name` - The database for which `dbschema` should generate definitions. If you omit `database_name`, `dbschema` uses the default database, if specified. (How you define the default database varies between operating systems. On UNIX, the value of the `DB_NAME` environment variable specifies the default database.)

See also:
• `Examples` (page 95)

### 6.1 Examples

The following example uses the `-t` option with a table list to generate the table definitions only for the specified table in the rdsdb database:

```bash
> dbschema -t dbp1,test_view rdsdb
    FairCom/DBSCHEMA Version 9.1.17573(Build-090721)

create table ADMIN.dbp1 (
    c1      integer
) pctfree 20;
create view ADMIN.test_view (  
    fld
) as
select * from test_revokel ;
```

The following example uses the `-p` option with the % wildcard character to generate definitions for all stored procedures whose names begin with the characters “foo”:

```bash
> dbschema -p foo% rdsdb
    FairCom/DBSCHEMA Version 9.1.17573(Build-090721)

create procedure ADMIN.foo (  
    p1 integer
) as
begin
end;
```
CREATE PROCEDURE ADMIN.foobar(in sno character (5),
in sname character (20),
in sstatus smallint,
in scity character (15))
IMPORT
BEGIN
SQLIStatement stmt = new SQLIStatement("insert into s values ('foo', 'foo', 3, '
foo')"); stmt.execute();
END

The following example uses the -o option to write all definitions for the rdsdb database to the file schema.sql:

```
> dbschema -o schema.sql -u ADMIN -a ADMIN rdsdb
FairCom/DBSCHEMA Version 9.1.17573(Build-090721)
FairCom Corporation (C) 1992-2009.

> more schema.sql
create table ADMIN.test_revoke1 (
    fld integer
) pctfree 20;
...
# 7. FairCom Typographical Conventions

Before you begin using this guide, be sure to review the relevant terms and typographical conventions used in the documentation.

The following formatted items identify special information.

<table>
<thead>
<tr>
<th>Formatting convention</th>
<th>Type of Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Used to emphasize a point or for variable expressions such as parameters</td>
</tr>
<tr>
<td><strong>CAPITALS</strong></td>
<td>Names of keys on the keyboard. For example, SHIFT, CTRL, or ALT+F4</td>
</tr>
</tbody>
</table>

*FairCom Terminology*

<table>
<thead>
<tr>
<th>FunctionName()</th>
<th>c-treeACE Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
<td>c-treeACE Function Parameter</td>
</tr>
<tr>
<td>Code Example</td>
<td>Code example or Command line usage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>utility</th>
<th>c-treeACE executable or utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>c-treeACE file or path name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONFIGURATION KEYWORD</th>
<th>c-treeACE Configuration Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTREE_ERR</td>
<td>c-treeACE Error Code</td>
</tr>
</tbody>
</table>
8. Index

@ (Execute) ............................................. 40
@ Execute syntax ................................... 40
A
Adding Beginning and Concluding Titles with the TITLE Statement .................................. 35
Adding titles ....................................... 35
Additional Resources .............................. 8, 15, 21, 27
B
Beginning titles .................................... 35
BREAK ............................................. 41
BREAK statement .................................. 30
BREAK statement syntax ......................... 41
C
CLEAR ............................................ 44
CLEAR statement .................................. 30
CLEAR statement syntax ......................... 44
COLUMN ........................................ 45
Column display formatting ........................ 32
COLUMN statement ............................... 30
COLUMN statement date-time formats ........... 45
COLUMN statement numeric formats .......... 45
COLUMN statement syntax ...................... 45
Commands file .................................... 81, 90
Compilation Errors ............................... 86
COMPUTE ...................................... 50
COMPUTE statement ............................. 30
COMPUTE statement syntax .................... 50
Concluding titles .................................. 35
Copyright Notice .................................. xcviii
D
Data Unload Utility .................................
dbdump ........................................... 88
Data File Formats ................................. 81, 90
Data file formats for dbdump ...................... 90
Data file formats for dbload ...................... 81
Data Load Tips .................................... 1
Data Load Utility .................................
dbload ............................................ 79
Data summaries ................................... 33
Date-time formats for COLUMN statement .... 45
dbdump commands file ............................ 90
data file formats ................................... 90
DEFINE RECORDS statement ................ 91
examples ........................................... 93
FOR RECORD statement ....................... 92
syntax ............................................ 88
dbload Command-Line Syntax ................... 88
dbload commands file ............................. 81
data file formats ................................... 81
DEFINE RECORDS statement ................ 82
errors ............................................... 85
examples .......................................... 84
FOR EACH statement ........................... 83
prerequisites ....................................... 80
syntax ............................................. 80
dbload Command Line Syntax .................. 80
dbload Errors ..................................... 85
dbschema examples ................................ 95
Define ............................................. 5, 12, 18, 24
DEFINE ......................................... 52
DEFINE RECORDS statement ............... 82, 91
DEFINE statement ............................... 30
DEFINE statement syntax ...................... 52
DISPLAY ........................................ 53
DISPLAY statement ............................. 30
DISPLAY statement syntax .................... 53
Done ............................................... 7, 14, 20, 26
E
EDIT ............................................. 55
EDIT statement .................................... 29
EDIT statement syntax ........................... 55
Errors dbload ...................................... 85
Examples ......................................... 84, 93, 95
EXIT or QUIT ..................................... 56
EXIT statement syntax ........................... 56
F
FairCom Typographical Conventions ........... 97
Fatal Errors ....................................... 86
FOR EACH statement ........................... 83
FOR RECORD statement ....................... 92
Formatting Column Display with the COLUMN Statement ............................................. 32
Formatting column displays ...................... 32
Formatting ISQL output .......................... 30
Formatting Output of ISQL Queries ............. 30
G
GET ............................................. 57
GET statement .................................... 29
GET statement syntax ........................... 57
H
HELP ............................................. 59
HELP statement syntax ........................... 59
HISTORY ......................................... 60
HISTORY statement .............................. 29
HISTORY statement syntax ..................... 60
HOST or SH or ! ................................. 61
HOST statement syntax .............................................61
I
IF [ NOT ] EXISTS .....................................................62
INDEXES ..................................................................63
Init ...........................................................................4, 11, 17, 23
Introduction ..................................................................1
Introductory Tutorial ..................................................3
ISQL
output formats ..........................................................30
reference .....................................................................39
starting ........................................................................28
statements for query formatting ..................................30
syntax ..........................................................................28
ISQL Reference ............................................................39
ISQL Statements ...........................................................28
L
LIST .............................................................................64
LIST statement ..........................................................29
LIST statement syntax ...............................................64
M
Manage ......................................................................6, 13, 19, 25
N
Numeric formats for COLUMN statement ..................45
O
Output formats ..........................................................30
P
Prerequisites for dbload .............................................80
Q
Queries, unformatted ................................................30
Quick Tour ..................................................................2
QUIT or EXIT ...........................................................65
QUIT statement syntax ...............................................56
R
Record/Row Locking ..................................................16
References for ISQL ....................................................39
Relationships .............................................................9
RUN .............................................................................66
RUN statement ..........................................................29
RUN statement syntax ...............................................66
S
SAVE ..........................................................................67
SAVE statement ........................................................29
SAVE statement syntax ...............................................67
Schema Export Utility
dbschema ..................................................................94
SET .............................................................................68
SET ECHO statement ..................................................30
SET LINESIZE statement ...........................................30
SET PAGESIZE statement ..........................................30
SET REPORT statement .............................................30
SET statement syntax .................................................68
SHOW .........................................................................71
SHOW statement syntax ............................................71
SPOOL .................................................................72
SPOOL statement syntax ............................................72
SQL Data Load Tips ......................................................1
START ..........................................................................73
START statement ......................................................29
START statement syntax ............................................73
Starting Interactive SQL ...............................................28
Starting ISQL ............................................................28
Statement history support ..........................................29
ISQL statements ..........................................................29
Statement History Support ...........................................29
Statements
@EXECUTE syntax .....................................................40
BREAK .................................................................30, 33, 41
BREAK syntax ..........................................................41
CLEAR .................................................................30, 44
CLEAR syntax ..........................................................44
COLUMN ...............................................................30, 32, 45
COLUMN date-time formats ........................................45
COLUMN numeric formats ..........................................45
COLUMN syntax ........................................................45
COMPUTE ..............................................................30, 33, 50
COMPUTE syntax ......................................................50
DEFINE .................................................................30, 52
DEFINE RECORD .....................................................82, 91
DEFINE syntax ..........................................................52
DISPLAY .................................................................30, 33, 53
DISPLAY syntax ........................................................53
EDIT .............................................................................29, 55
EDIT syntax ..............................................................55
EXIT .............................................................................56
EXIT syntax ..............................................................56
FOR EACH ...............................................................83
FOR RECORD ...........................................................92
GET .............................................................................29, 57
GET syntax ..............................................................57
HELP ............................................................................36, 59
HELP syntax ..............................................................59
HISTORY .................................................................29, 60
HISTORY syntax ........................................................60
HOST ..........................................................................61
HOST syntax ............................................................61
LIST .............................................................................29, 64
LIST syntax ..............................................................64
QUIT .............................................................................56
QUIT syntax ..............................................................56
RUN .............................................................................29, 66
RUN syntax ..............................................................66
SAVE ..........................................................................29, 67
SAVE syntax ..............................................................67
SET .............................................................................68
SET .............................................................................30
SET ECHO ..................................................................30
SET LINESIZE .........................................................30
Index

SET PAGESIZE ...........................................30
SET REPORT ...........................................30
SET syntax ............................................68
SHOW ..................................................71
SHOW syntax ........................................71
SPOOL ..................................................72
SPOOL syntax ........................................72
START ..................................................29, 73
START syntax ........................................73
TABLE ..................................................36, 75
TABLE syntax ........................................75
TITLE ..................................................30, 77
TITLE syntax ........................................77
TITLES ..................................................35
Statements for query formatting .........................30
Summarizing data ......................................33
Summarizing Data with DISPLAY, COMPUTE, and BREAK Statements ........................................33
Syntax for ISQL .........................................28

T
TABLE ..................................................75
TABLE statement syntax ................................75
The Commands File ....................................81, 90
The DEFINE RECORD Statement ......................82, 91
The FOR EACH Statement .................................83
The FOR RECORD Statement ..............................92
The HELP and TABLE Statements .......................36
TITLE ..................................................77
TITLE statement ........................................30
TITLE statement syntax ................................77
Titles
adding ...................................................35
beginning ................................................35
concluding ..............................................35
Transaction Processing ..................................22
Transaction support .....................................37
Transaction Support .................................37

U
Unformatted queries ...................................30

W
Work Flow Control ....................................37