

Importing and Exporting Data

About this chapter

Transferring large amounts of data into and out of your database may be necessary in several situations. For example:

- ◆ Importing an initial set of data into a new database
- ◆ Exporting data into other applications, such as spreadsheets
- ◆ Building new copies of a database with a modified structure
- ◆ Creating extractions of a database

This chapter describes how to import data to and export data from databases, both in text form and in other formats.

Contents

Topic	Page
Import and export overview	278
Exporting data from a database	280
Importing data into a database	285

Import and export overview

Adaptive Server Anywhere supports import and export of individual tables or a complete database using text files. Importing and exporting data to other formats, such as spreadsheet program formats, is available from the Interactive SQL utility.

The LOAD TABLE statement and the UNLOAD TABLE statement are the SQL statements for loading and unloading tables using text files. The INPUT statement and the OUTPUT statement are Interactive SQL commands for importing and exporting data using a variety of formats.

You can unload individual tables or a complete database from Sybase Central, using the Unload Database wizard. You can also unload individual tables or a complete database using the DBUNLOAD command line utility. The DBUNLOAD utility is accessible from Interactive SQL.

This chapter describes the LOAD TABLE and UNLOAD TABLE SQL statements, as well as the use of Sybase Central, DBUNLOAD, and Interactive SQL to import and export data.

Input and output data formats

The LOAD TABLE and UNLOAD TABLE import and export text files, with one row per line, and values separated by a delimiter.

The Interactive SQL INPUT and OUTPUT statements support the following file formats:

File Format	Description
ASCII	A text file, one row per line, with values separated by a delimiter. String values are optionally enclosed in apostrophes (single quotes). This is the same as the format used by LOAD TABLE and UNLOAD TABLE
DBASEII	DBASE II format
DBASEIII	DBASE III format
DIF	Data Interchange Format
FIXED	Data records are in fixed format with the width of each column either the same as defined by the column's type or specified as a parameter
FOXPRO	FoxPro format
LOTUS	Lotus workspace format
WATFILE	WATFILE format.

There is a Interactive SQL option for the default input format and one for the default output format. For details on how to specify these options, see "INPUT_FORMAT option" on page 157 of the book *Adaptive Server Anywhere Reference Manual* and "OUTPUT_FORMAT option" on page 166 of the book *Adaptive Server Anywhere Reference Manual*.

The file format can alternatively be specified on each INPUT statement or OUTPUT statement.

Exporting data from a database

You can export data from your database using one of the following methods:

- ◆ The UNLOAD TABLE statement, for efficient export of text files
- ◆ The Interactive SQL OUTPUT statement, for slower but more flexible export to a variety of file formats
- ◆ The DBUNLOAD utility, for text export of more than one table

This section describes each of these methods, and also describes some tips for dealing with NULL output.

Unloading data using the UNLOAD TABLE statement

The UNLOAD TABLE statement is used for efficient export of data from a database table to a text file. You must have SELECT permission on the table to use the UNLOAD TABLE statement.

Example 1

The following statement unloads the **department** table from the sample database into the file *dept.txt* in the server's current working directory. If you are running against a network server, the command unloads the data into a file on the server machine, not the client machine.

```
UNLOAD TABLE department
TO 'dept.txt'
```

The *dept.txt* file has the following contents:

```
100,'R & D',501
200,'Sales',902
300,'Finance',1293
400,'Marketing',1576
500,'Shipping',703
```

Notes

- ◆ Each row of the table is output on a single line of the output file.
- ◆ No column names are exported.
- ◆ The columns are separated, or *delimited*, by a comma. The delimiter character can be changed using the DELIMITED BY clause.
- ◆ The fields are not fixed-width fields. Only the characters in each entry are exported, not the full width of the column.
- ◆ The character data in the **dept_name** column is enclosed in single quotes. The single quotes can be turned off using the QUOTES clause.

- ◆ The data is exported in order by primary key values. This makes reloading quicker. You can export data in the order in which it is stored using the ORDER OFF clause.
- ◆ The file name is relative to the server's current directory, not the current directory of the client application. Also, the file name is passed to the server as a string. It is recommended that you escape backslash characters in the file name to prevent misinterpretation if a directory of file name begins with an n (\n is a newline character). For example, the following statement unloads a table into the file *c:\temp\newfile.dat*:

```
UNLOAD TABLE employee
TO 'c:\\temp\\newfile.dat'
```

For more information on the syntax, see "UNLOAD TABLE statement" on page 570 of the book *Adaptive Server Anywhere Reference Manual*.

Example 2

The following example uses explicit settings for the DELIMITED BY and QUOTES clauses:

```
UNLOAD TABLE department
TO 'dept.txt'
DELIMITED BY '$'
QUOTES OFF
```

The resulting *dept.txt* file has the following contents:

```
100$R & D$501
200$Sales$902
300$Finance$1293
400$Marketing$1576
500$Shipping$703
```

If a delimiter character appears within a value and the QUOTES option is turned off, the character is replaced by its hexadecimal value preceded by \x. For example:

```
UNLOAD TABLE department
TO 'dept.txt'
DELIMITED BY '&'
QUOTES OFF
```

yields the following output file.

```
100&R \x26 D&501
200&Sales&902
300&Finance&1293
400&Marketing&1576
500&Shipping&703
```

Exporting query results using Interactive SQL

You can export queries to a file from Interactive SQL either by using the OUTPUT statement or by redirecting output.

Using the output statement

Data can be exported from a database to a variety of file formats using the Interactive SQL OUTPUT statement. This statement exports the results of the current query (the one displayed in the Interactive SQL Data window) and puts the results into a specified file. The output format can be specified on the output command.

For example, the following commands extract the **employee** table to a dBaseIII format file:

```
SELECT *
FROM employee;

OUTPUT TO employee.dbf
FORMAT dbaseiii;
```

Using output redirection

Output redirection can be used to export data as an alternative to the OUTPUT statement.

The output of any command can be redirected to a file or device by putting the **>#** redirection symbol anywhere on the command. The redirection symbol must be followed by a file name, as follows:

```
SELECT *
FROM employee
># filename
```

Do not enclose the file name in quotation marks.

Output redirection is most useful on the SELECT statement. The SET OUTPUT_FORMAT command can be used to control the format of the output file.

Redirecting with error messages

The **>&** redirection symbol redirects all output including error messages and statistics for the command on which it appears. For example:

```
SELECT *
FROM employee
>& filename
```

outputs the SELECT statement to the file, followed by the output from the SELECT statement and some statistics pertaining to the command.

Do not enclose the file name in quotation marks.


The `>&` redirection is useful for getting a log of what happens during a `READ` command. The statistics and errors of each command are written following the command in the redirected output file.

If two `>` characters are used in a redirection symbol instead of one (`>>#` or `>>&.`), the output is appended to the specified file instead of replacing the contents of the file. For output from the `SELECT` statement, headings are output if and only if the output starts at the beginning of the specified file and the output format supports headings.

NULL value output

The most common reason to extract data is for use in other software products. The other software package may not understand `NULL` values.

There is a Interactive SQL option (`NULLS`) that allows you to choose how `NULL` values are output. Alternatively, you can use the `IFNULL` function to output a specific value whenever there is a `NULL` value.


 For information on setting Interactive SQL options, see "SET OPTION statement" on page 553 of the book *Adaptive Server Anywhere Reference Manual*.

Unloading a database using DBUNLOAD

The *dbunload* utility is used to unload an entire database in ASCII comma-delimited format and to create the necessary Interactive SQL command files to completely recreate your database. This may be useful for creating extractions, creating a backup of your database, or building new copies of your database with the same or a slightly modified structure.

If you want to rearrange your tables in the database, you can use *dbunload* to create the necessary command files and modify them as needed.

For Windows 3.x, the `DBUNLOAD` executable is named *dbunloaw.exe*.

 For a full description of *dbunload* utility command-line switches, see the section "The Unload utility" on page 110 of the book *Adaptive Server Anywhere Reference Manual*.

Exporting a list of tables

The *dbunload* utility can also export a list of tables, rather than the entire database. This is useful for retrieving data from a corrupted database that cannot be entirely unloaded.

The following statement unloads the data from the sample database (assumed to be running on the default database server with the default database name) into a set of files in the *c:\temp* directory. A command file to rebuild the database from the data files is created with the default name *reload.sql* in the current directory.

```
dbunload -c "dbn=asademo;uid=dba;pwd=sql" c:\temp
```


Importing data into a database

You can import data into your database using one of the following methods:

- ◆ The LOAD TABLE statement, for efficient import of text files.
- ◆ The Interactive SQL INPUT statement, for slower but more flexible import of a variety of file formats.
- ◆ Interactive input using the INSERT Statement or the Interactive SQL INPUT statement.

This section describes each of these methods, and also describes some tips for dealing with incompatible data structure and conversion errors.

Loading data using the LOAD TABLE statement

The LOAD TABLE statement is used for efficient importing of data from a text file into a database table. The table must exist and have the same number of columns as the input file has fields, defined on compatible data types. In order to use the LOAD TABLE statement, the user must have INSERT permission on the table.

Example


If the **department** table had all its rows deleted, the following statement would load the data from the file *dept.txt* into the department table:

```
LOAD TABLE department
FROM 'dept.txt'
```

The LOAD TABLE statement appends the contents of the file to the existing rows of the table; it does not replace the existing rows in the table. You can use the TRUNCATE TABLE statement to remove all the rows from a table.

Neither the TRUNCATE TABLE statement nor the LOAD TABLE statement fires triggers, including referential integrity actions such as cascaded deletes.

The LOAD TABLE statement has many of the same options as the UNLOAD TABLE statement.

 For a description of column delimiters, use of quotes, and file names, see "Unloading data using the UNLOAD TABLE statement" on page 280.

The LOAD TABLE statement has the additional STRIP clause. The default setting (STRIP ON) strips trailing blanks from values before they are inserted. To keep trailing blanks, use the STRIP OFF clause in your LOAD TABLE statement.

☞ For a full description of the LOAD TABLE statement syntax, see "LOAD TABLE statement" on page 504 of the book *Adaptive Server Anywhere Reference Manual*.

Importing data using the Interactive SQL INPUT statement

Data with the same structure as existing database tables can be loaded into your database from a file using the Interactive SQL INPUT statement.

The Interactive SQL INPUT statement is less efficient than the LOAD TABLE statement for importing text files. However, the INPUT statement supports several different file formats, whereas the LOAD TABLE statement can be used only for text files.

The INPUT command can be entered in Interactive SQL as follows:

```
INPUT INTO t1
FROM file1
FORMAT ASCII;

INPUT INTO t2
FROM file2
FORMAT FIXED
COLUMN WIDTHS (5, 10, 40, 40 );
...
```

These statements could be put in a command file which can then be executed in Interactive SQL for modification and reference.

☞ For more information about command files, see the tutorial chapter "Running command files" on page 80 of the book *First Guide to SQL Anywhere Studio*.

Loading data interactively

There are two commands that can be used to input data interactively. You can use the insert command:

```
INSERT INTO T1
VALUES ( ... )
```

to insert a single row at a time or you can use the input command:

```
INPUT INTO T1 PROMPT
```

which gives you a full screen to type in data in the current input format (controlled by the Interactive SQL INPUT_FORMAT option).

Handling conversion errors on data import

When you are loading data from external sources, there may be errors in the data. For example, there may be dates that are not valid dates and numbers that are not valid numbers. The `CONVERSION_ERROR` database option allows you to ignore conversion errors by converting them to `NULL` values.

For information on setting Interactive SQL database options, see "SET OPTION statement" on page 553 of the book *Adaptive Server Anywhere Reference Manual*.

Loading data that does not match the table structure

The structure of the data to be loaded into a table does not always match the structure of the destination table itself. For example, the column data types may be different, or in different order, or there may be extra values in the data to be imported that do not match columns in the destination table.

❖ To load data that has a different structure:

- 1 Using the `LOAD TABLE` statement, load the data into a temporary table that has a structure matching that of the input file.
- 2 Use the `INSERT` statement with a `FROM SELECT` clause to extract and summarize data from the temporary table and put it into one or more of the permanent database tables.

`DECLARE
TEMPORARY
TABLE` statement

If you are loading a set of data once and for all, you should make the temporary table using the `DECLARE TEMPORARY TABLE` statement. A declared temporary table exists only for the duration of a connection or, if defined inside a compound statement, of the compound statement.

`CREATE TABLE`
statement

If you are loading data of a similar structure repeatedly, you should make the temporary table using the `CREATE TABLE` statement, specifying a global temporary table. The definition of the temporary table is held in the database permanently, but the rows exist only within a given connection.

Tuning bulk loading of data

Loading large volumes of data into a database can be very time consuming. There are a few things you can do to save time.

- ◆ If you are using the `INPUT` command, run Interactive SQL or the client application on the same machine as the server. Loading data over the network adds extra communication overhead. This might mean loading new data during off hours.

- ◆ Place data files on a separate physical disk drive from the database file. This could avoid excessive disk head movement during the load.
- ◆ Increase the size of the database cache. Eliminate disk cache in favor of database cache if the machine is a dedicated database server.

☞ For a description of how to control the cache size from the server command line option see "The database server" on page 12 of the book *Adaptive Server Anywhere Reference Manual*.

- ◆ If you are using the INPUT command, start the server with the `-b` switch for bulk operations mode. In this mode, the server does not keep a rollback log or a transaction log, it does not perform an automatic COMMIT before data definition commands, and it does not lock any records.

Without a rollback log, you cannot use savepoints and aborting a command always causes transactions to roll back. Without automatic COMMIT, a ROLLBACK undoes everything since the last explicit COMMIT.

Without a transaction log, there is no log of the changes. You should back up the database file before and after using bulk operations mode because, in this mode, your database is not protected against media failure. For more information, see "Backup and Data Recovery" on page 553.

The server allows only one connection when you use the `-b` switch.

If you have data that requires many commits, running with the `-b` option may slow database operation. At each COMMIT, the server carries out a checkpoint; this frequent checkpointing can slow the server.

- ◆ Extend the size of the database file, as described in "ALTER DBSPACE statement" on page 345 of the book *Adaptive Server Anywhere Reference Manual*. This command allows a database file to be extended in large amounts before the space is required, rather than the normal 32 pages at a time when the space is needed. As well as improving performance for loading large amounts of data, it also serves to keep the database files more contiguous within the file system.