CHAPTER 12

Selecting Data from Database Tables

About this chapter

This tutorial introduces the SELECT statement, the statement you use to retrieve information from databases. SELECT statements are commonly called **queries**, because they ask the database engine about the information in a database.

Contents

Before you begin the tutorial

The SELECT statement is a versatile command. SELECT statements can become highly complex in applications retrieving very specific information from large databases. This tutorial uses only simple SELECT statements: more advanced queries are described in later tutorials.

For more information about the full syntax of the select statement, see "SELECT statement" on page 542 of the book *Adaptive Server Anywhere Reference Manual*.

Notes

Ideally, you should be running the Adaptive Server Anywhere software on your computer while you read and work through the tutorial lessons.

Each lesson instructs you to type commands into the computer and describes what you will see on your computer screen. If you cannot run the software as you read the tutorials, you will still be able to learn about SQL but you will not have the opportunity to experiment on your own.

This tutorial assumes that you have already started Interactive SQL and connected to the sample database. If you have not already done so, see "Using Interactive SQL" on page 71.

Looking at the information in a table

The database you use in this tutorial is for a fictional company. The database contains information about employees, departments, sales orders, and so on. All the information is organized into a number of **tables**.

Using the SELECT statement

In this lesson, you look at one of the tables in the database. The command used will look at everything in a table called **employee**. Execute the command:

```
SELECT * FROM employee
```

Case sensitivity

The table name **employee** is shown starting with an upper case E, even though the real table name is all lower case. Adaptive Server Anywhere databases can be created as case-sensitive or case-sensitive in their string comparisons, but are always case insensitive in their use of identifiers.

For information on creating databases, see "Initializing a database" on page 68 of the book *Adaptive Server Anywhere User's Guide*, or "The Initialization utility" on page 84 of the book *Adaptive Server Anywhere Reference Manual*.

You can type **select** or **Select** instead of **SELECT**. You can enter SQL keywords in upper case, lower case, or any combination of the two. In the manuals, upper case letters are generally used for SQL keywords.

The SELECT statement retrieves all the rows and columns of the **employee** table, and the Interactive SQL Data window lists those that will fit:

emp_id	manager_id	emp_fname	emp_lname	dept_id
102	501	Fran	Whitney	100
105	501	Matthew	Cobb	100
129	902	Philip	Chin	200
148	1293	Julie	Jordan	300
160	501	Robert	Breault	100

You will also see some information in the Interactive SQL statistics window. This information is explained later.

The **employee** table contains a number of **rows** organized into **columns**. Each column has a name, such as **emp_lname** or **emp_id**. There is a row for each employee of the company, and each row has a value in each column. For example, the employee with employee ID 102 is Fran Whitney. Her manager has employee ID 501.

Manipulation of the Interactive SQL environment is specific to the operating system you are running in.

Ordering query results

Unless otherwise requested, the database server returns the rows of a table in no particular order. Often it is useful to look at the rows in a table in a more meaningful sequence. For example, you might like to see employees in alphabetical order.

The following example shows how adding an ORDER BY clause to the SELECT statement causes the results to be retrieved in alphabetical order.

❖ **To list the employees in alphabetical order:**

♦ Type the following:

```
SELECT * FROM employee ORDER BY emp_lname
```

emp_id	manager_id	emp_fname	emp_lname	dept_id
1751	1576	Alex	Ahmed	400
1013	703	Joseph	Barker	500
591	1576	Irene	Barletta	400
191	703	Jeannette	Bertrand	500
1336	1293	Janet	Bigelow	300

Notes

The order of the clauses is important. The ORDER BY clause must follow the FROM clause and the SELECT clause.

Selecting columns from a table

Often, you are only interested in some of the columns in a table. For example, to make up birthday cards for employees you might want to see the **emp_lname**, **dept_id**, and **birth_date** columns.

❖ **List the last name, department, and birthdate of each employee:**

♦ Type the following:

```
SELECT emp_lname, dept_id, birth_date
FROM employee
```

emp_lname	dept_id	birth_date
Whitney	100	1958-06-05
Cobb	100	1960-12-04
Chin	200	1966-10-30
Jordan	300	1951-12-13
Breault	100	1947-05-13

Rearranging columns

The three columns appear in the order in which you typed them in the SELECT command. If you want to rearrange the columns, simply change the order of the column names in the command. For example, to put the **birth_date** column on the left, use the following command:

```
SELECT birth_date, emp_lname , dept_id
FROM employee
```

Ordering rows

You can order rows and look at only certain columns at the same time as follows:

```
SELECT birth_date, emp_lname , dept_id
FROM employee
ORDER BY emp_lname
```

As you might have guessed, the asterisk in

```
SELECT * FROM employee
```

is a short form for all columns in the table.

Selecting rows from a table

Sometimes you will not want to see information on all the employees in the **employee** table. Adding a WHERE clause to the SELECT statement allows only some rows to be selected from a table.

For example, suppose you would like to look at the employees with first name **John**.

❖ **List all employees named John:**

♦ Type the following:

```
SELECT *
FROM employee
WHERE emp_fname = 'John'
```

emp_id	manager_id	emp_fname	emp_lname	dept_id
318	1576	John	Crow	400
862	501	John	Sheffield	100
1483	1293	John	Letiecq	300

Apostrophes and case-sensitivity

♦ The apostrophes (single quotes) around the name 'John' are required. They indicate that **John** is a character string. Quotation marks (double quotes) have a different meaning. Quotation marks can be used to make otherwise invalid strings valid for column names and other identifiers.

♦ The sample database is not case sensitive, so you would get the same results whether you searched for ' 'JOHN', 'john', or 'John'.

Again, you can combine what you have learned:

```
SELECT emp_fname, emp_lname, birth_date
FROM employee
WHERE emp_fname = 'John'
ORDER BY birth_date
```

Notes

♦ How you order clauses is important. The FROM clause comes first, followed by the WHERE clause, and then the ORDER BY clause. If you type the clauses in a different order, you will get a syntax error.

♦ You do not need to split the statement into several lines. You can enter the statement into the command window in any format. If you use more than the number of lines that fit on the screen, the text scrolls in the Command window.

Comparing dates in queries

Sometimes, you will not know exactly what value you are looking for, or you would like to see a set of values. You can use comparisons in the WHERE clause to select a set of rows that satisfy the search condition. The following example shows the use of a date inequality search condition.

❖ **To list all employees born before March 3, 1964"**

♦ Type the following:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date < 'March 3, 1964'
```

emp_lname	birth_date
Whitney	1958-06-05
Cobb	1960-12-04
Jordan	1951-12-13
Breault	1947-05-13
Espinoza	1939-12-14
Dill	1963-07-19

The database server knows that the **birth_date** column contains a date, and converts *'March 3, 1964'* to a date automatically.

Compound search conditions in the WHERE clause

So far, you have seen equal (=) and less than (<) as comparison operators. Adaptive Server Anywhere also supports other comparison operators, such as greater than (>), greater than or equal (>=), less than or equal (<=), and not equal (<>).

These conditions can be combined using AND and OR to make more complicated search conditions.

❖ **To list all employees born before March 3, 1964, but exclude the employee named Whitney:**

♦ Type the following:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date < '1964-3-3'
AND emp_lname <> 'whitney'
```

emp_lname	birth_date
Cobb	1960-12-04
Jordan	1951-12-13
Breault	1947-05-13
Espinoza	1939-12-14
Dill	1963-07-19
Francis	1954-09-12

Pattern matching in search conditions

Another useful way to look for things is to search for a pattern. In SQL, the word LIKE is used to search for patterns. The use of LIKE can be explained by example.

❖ **To list all employees whose last name begins with BR:**

♦ Type the following:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE emp_lname LIKE 'br%'
```

emp_lname	emp_fname
Breault	Robert
Braun	Jane

The *%* in the search condition indicates that any number of other characters may follow the letters **BR**.

❖ **To list all employees whose surname begins with** BR**, followed by zero or more letters and a** T**, followed by zero or more letters:**

♦ Type the following:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE emp_lname LIKE 'BR%T%'
```

emp_lname	emp_fname
Breault	Robert

The first % sign matches the string *eaul*, while the second % sign matches the empty string (no characters).

Another special character that can be used with LIKE is the _ (underscore) character, which matches exactly one character.

The pattern **BR_U%** matches all names starting with **BR** and having **U** as the fourth letter. In **Braun** the **_** matches the letter **A** and the **%** matches **N**.

Matching rows by sound

With the SOUNDEX function, you can match rows by sound, as well as by spelling. For example, suppose a phone message was left for a name that sounded like "Ms. Brown". Which employees in the company have names that sound like Brown?

❖ **To list employees with surnames that sound like Brown:**

♦ Type the following:

```
SELECT emp_lname, emp_fname
FROM employee
WHERE SOUNDEX( emp_lname ) = SOUNDEX( 'Brown' )
```

emp_lname	emp_fname
Braun	Jane

Jane Braun is the only employee matching the search condition.

Short cuts for typing search conditions

Using the shortform BETWEEN

SQL has two short forms for typing in search conditions. The first, BETWEEN, is used when you are looking for a range of values. For example the following query:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date BETWEEN '1963-1-1'
AND '1965-3-31'
```

is equivalent to the following:

```
SELECT emp_lname, birth_date
FROM employee
WHERE birth_date >= '1963-1-1'
AND birth_date <= '1965-3-31'
```

Using the short form IN

The second short form, IN, may be used when looking for one of a number of values. The command

```
SELECT emp_lname, emp_id
FROM employee
WHERE emp_lname IN ('yeung','bucceri','charlton')
```

means the same as:

```
SELECT emp_lname, emp_id
FROM employee
WHERE emp_lname = 'yeung'
OR emp_lname = 'bucceri'
OR emp_lname = 'charlton'
```