Introduction to the SELECT Statement

The SELECT statement, the command that represents Data Query Language (*DQL*) in SQL, is the statement used to construct database queries. The SELECT statement is not a standalone statement, which means that one or more additional clauses (elements) are required for a syntactically correct query. In addition to the required clauses, there are optional clauses that increase the overall functionality of the SELECT statement. The SELECT statement is by far one of the most powerful statements in SQL. The FROM clause is the mandatory clause and must always be used in conjunction with the SELECT statement.

NEW TERM There are four keywords, or *clauses,* that are valuable parts of a SELECT statement. These keywords are as follows:

- SELECT
- FROM
- WHERE
- ORDER BY

Each of these keywords is covered in detail during the following sections.

The SELECT Statement

The SELECT statement is used in conjunction with the FROM clause to extract data from the database in an organized, readable format. The SELECT part of the query is for selecting the data you want to see according to the columns in which they are stored in a table.

The syntax for a simple SELECT statement is as follows:

SYNTAX

```
SELECT [ * | ALL | DISTINCT COLUMN1, COLUMN2 ]
FROM TABLE1 [ , TABLE2 ];
```

The SELECT keyword in a query is followed by a list of columns that you want displayed as part of the query output. The FROM keyword is followed by a list of one or more tables from which you want to select data. The asterisk (*) is used to denote that all columns in a table should be displayed as part of the output. Check your particular implementation for its usage. The ALL option is used to display all values for a column, including duplicates. The DISTINCT option is used to suppress duplicate rows from being displayed in the output. The default between DISTINCT and ALL is ALL, which does not have to be specified. Notice that the columns following the SELECT are separated by commas, as is the table list following the FROM.

Note

Commas are used to separate arguments in a list in SQL statements. Some common lists include lists of columns in a query, lists of tables to be selected from in a query, values to be inserted into a table, and values grouped as a condition in a query's WHERE clause.

NEW TERM *Arguments* are values that are either required or optional to the syntax of a SQL statement or command.

Explore the basic capabilities of the SELECT statement by studying the following examples. First, perform a simple query from the PRODUCTS_TBL table:

INPUT

```
SELECT * FROM PRODUCTS_TBL;
```

OUTPUT

```
PROD_ID    PROD_DESC                      COST
---------- ------------------------------ ------
11235      WITCHES COSTUME                29.99
222        PLASTIC PUMPKIN 18 INCH         7.75
13         FALSE PARAFFIN TEETH            1.1
90         LIGHTED LANTERNS               14.5
15         ASSORTED COSTUMES              10
9          CANDY CORN                      1.35
6          PUMPKIN CANDY                   1.45
87         PLASTIC SPIDERS                 1.05
119        ASSORTED MASKS                  4.95
1234       KEY CHAIN                       5.95
2345       OAK BOOKSHELF                  59.99

11 rows selected.
```

The asterisk represents all columns in the table, which, as you can see, are displayed in the form PROD_ID, PROD_DESC, and COST. Each column in the output is displayed in the order that it appears in the table. There are 11 records in this table, identified by the feedback `11 rows selected`. This feedback differs among implementations; for example, another feedback for the same query would be `11 rows affected`.

Now select data from another table, CANDY_TBL. Create this table in the image of the PRODUCTS_TBL table for the following examples. List the column name after the SELECT keyword to display only one column in the table:

INPUT

```
SELECT PROD_DESC FROM CANDY_TBL;
```

OUTPUT

```
PROD_DESC
------------------
CANDY CORN
CANDY CORN
HERSHEYS KISS
SMARTIES

4 rows selected.
```

Four records exist in the CANDY_TBL table. You have used the ALL option in the next statement to show you that the ALL is optional and redundant. There is never a need to specify ALL; it is a default option.

INPUT

```
SELECT ALL PROD_DESC

FROM CANDY_TBL;
```

OUTPUT

```
PROD_DESC
-------------------
CANDY CORN
CANDY CORN
HERSHEYS KISS
SMARTIES

4 rows selected.
```

The DISTINCT option is used in the following statement to suppress the display of duplicate records. Notice that the value CANDY CORN is only printed once in this example.

INPUT

```
SELECT DISTINCT PROD_DESC

FROM CANDY_TBL;
```

OUTPUT

```
PROD_DESC
------------------
CANDY CORN
HERSHEYS KISS
SMARTIES
```

```
3 rows selected.
```

DISTINCT and ALL can also be used with parentheses enclosing the associated column. The use of parentheses is often used in SQL—as well as many other languages—to improve readability.

INPUT

```
SELECT DISTINCT(PROD_DESC)

FROM CANDY_TBL;
```

OUTPUT

```
PROD_DESC
------------------
CANDY CORN
HERSHEYS KISS
SMARTIES

3 rows selected.
```

The FROM Clause

The FROM clause must be used in conjunction with the SELECT statement. It is a required element for any query. The FROM clause's purpose is to tell the database what table(s) to access to retrieve the desired data for the query. The FROM clause may contain one or more tables. The FROM clause must always list at least one table.

The syntax for the FROM clause is as follows:

SYNTAX

```
FROM TABLE1 [ , TABLE2 ]
```

Using Conditions to Distinguish Data

NEW TERM

A *condition* is part of a query that is used to display selective information as specified by the user. The value of a condition is either TRUE or FALSE, thereby limiting the data received from the query. The WHERE clause is used to place conditions on a query by eliminating rows that would normally be returned by a query without conditions.

There can be more than one condition in the WHERE clause. If there is more than one condition, they are connected by the AND and OR operators, which are discussed during Hour 8, "Using Operators to Categorize Data." As you also learn during the next hour, there are several conditional operators that can be used to specify conditions in a query. This hour only deals with a single condition for each query.

NEW TERM An *operator* is a character or keyword in SQL that is used to combine elements in a SQL statement.

The syntax for the WHERE clause is as follows:

SYNTAX

```
SELECT [ ALL | * | DISTINCT COLUMN1, COLUMN2 ]
FROM TABLE1 [ , TABLE2 ]
WHERE [ CONDITION1 | EXPRESSION1 ]
[ AND CONDITION2 | EXPRESSION2 ]
```

The following is a simple SELECT without conditions specified by the WHERE clause:

INPUT

```
SELECT *

FROM PRODUCTS_TBL;
```

OUTPUT

```
PROD_ID    PROD_DESC                    COST
---------- ---------------------------- ------
11235      WITCHES COSTUME              29.99
222        PLASTIC PUMPKIN 18 INCH       7.75
13         FALSE PARAFFIN TEETH          1.1
90         LIGHTED LANTERNS             14.5
15         ASSORTED COSTUMES            10
9          CANDY CORN                    1.35
6          PUMPKIN CANDY                 1.45
87         PLASTIC SPIDERS               1.05
119        ASSORTED MASKS                4.95
1234       KEY CHAIN                     5.95
2345       OAK BOOKSHELF                59.99

11 rows selected.
```

Now add a condition for the same query.

INPUT

```
SELECT * FROM PRODUCTS_TBL

WHERE COST < 5;
```

OUTPUT

```
PROD_ID    PROD_DESC                      COST
---------- ------------------------------ -----
13         FALSE PARAFFIN TEETH           1.1
9          CANDY CORN                     1.35
6          PUMPKIN CANDY                  1.45
87         PLASTIC SPIDERS                1.05
119        ASSORTED MASKS                 4.95

5 rows selected.
```

The only records displayed are those that cost less than $5.

In the following query, you want to display the product description and cost that matches the product identification 119.

INPUT

```
SELECT PROD_DESC, COST

FROM PRODUCTS_TBL

WHERE PROD_ID = '119';
```

OUTPUT

```
PROD_DESC                      COST
------------------------------ -----
ASSORTED MASKS                 4.95

1 row selected.
```

Sorting Your Output

You usually want your output to have some kind of order. Data can be sorted by using the ORDER BY clause. The ORDER BY clause arranges the results of a query in a listing format you specify. The default ordering of the ORDER BY clause is an *ascending order;* the sort displays in the order A–Z if it's sorting output names alphabetically. A *descending order* for alphabetical output would be displayed in the order Z–A. Ascending order for output for numeric values between 1 and 9 would be displayed 1–9; descending order is displayed as 9–1.

Note

SQL sorts are *ASCII*, character-based sorts. The numeric values 0–9 would be sorted as character values, and sorted before the characters A–Z. Because numeric values are treated like characters during a sort, the following list of numeric values would be sorted in the following order: 1, 12, 2, 255, 3.

The syntax for the ORDER BY is as follows:

SYNTAX

```
SELECT [ ALL | * | DISTINCT COLUMN1, COLUMN2 ]
FROM TABLE1 [ , TABLE2 ]
WHERE [ CONDITION1 | EXPRESSION1 ]
[ AND CONDITION2 | EXPRESSION2 ]
ORDER BY COLUMN1|INTEGER [ ASC|DESC ]
```

Begin your exploration of the ORDER BY clause with an extension of one of the previous statements. Order by the product description in ascending order or alphabetical order. Note the use of the ASC option. ASC can be specified after any column in the ORDER BY clause.

INPUT

```
SELECT PROD_DESC, PROD_ID, COST

FROM PRODUCTS_TBL

WHERE COST < 20

ORDER BY PROD_DESC ASC;
```

OUTPUT

```
PROD_DESC                 PROD_ID          COST
------------------------- --------------- ------
ASSORTED COSTUMES         15              10
ASSORTED MASKS            119              4.95
CANDY CORN                9                1.35
FALSE PARAFFIN TEETH      13               1.1
LIGHTED LANTERNS          90              14.5
PLASTIC PUMPKIN 18 INCH   222              7.75
PLASTIC SPIDERS           87               1.05
PUMPKIN CANDY             6                1.45

8 rows selected.
```

Note

Because ascending order for output is the default, ASC does not have to be specified.

You can use DESC, as in the following statement, if you want the same output to be sorted in reverse alphabetical order.

INPUT

```
SELECT PROD_DESC, PROD_ID, COST
```

```
FROM PRODUCTS_TBL

WHERE COST < 20

ORDER BY PROD_DESC DESC;
```

OUTPUT

```
PROD_DESC                 PROD_ID          COST
------------------------- -------------- ------
PUMPKIN CANDY             6               1.45
PLASTIC SPIDERS           87              1.05
PLASTIC PUMPKIN 18 INCH   222             7.75
LIGHTED LANTERNS          90             14.5
FALSE PARAFFIN TEETH      13              1.1
CANDY CORN                9               1.35
ASSORTED MASKS            119             4.95
ASSORTED COSTUMES         15             10

8 rows selected.
```

There are shortcuts in SQL. A column listed in the ORDER BY clause can be abbreviated with an integer. The INTEGER is a substitution for the actual column name (an alias for the purpose of the sort operation), identifying the position of the column after the SELECT keyword.

An example of using an integer as an identifier in the ORDER BY clause follows:

INPUT

```
SELECT PROD_DESC, PROD_ID, COST

FROM PRODUCTS_TBL

WHERE COST < 20

ORDER BY 1;
```

OUTPUT

```
PROD_DESC                 PROD_ID          COST
------------------------- -------------- ------
ASSORTED COSTUMES         15             10
ASSORTED MASKS            119             4.95
CANDY CORN                9               1.35
FALSE PARAFFIN TEETH      13              1.1
LIGHTED LANTERNS          90             14.5
PLASTIC PUMPKIN 18 INCH   222             7.75
PLASTIC SPIDERS           87              1.05
PUMPKIN CANDY             6               1.45
```

```
8 rows selected.
```

In this query, the integer 1 represents the column PROD_DESC. The integer 2 represents the PROD_ID column, 3 represents the COST column, and so on.

You can order by multiple columns in a query, using either the column name itself or the associated number of the column in the SELECT:

```
ORDER BY 1,2,3
```

Columns in an ORDER BY clause are not required to appear in the same order as the associated columns following the SELECT, as shown by the following example:

```
ORDER BY 1,3,2
```

Case Sensitivity

Case sensitivity is a very important concept to understand when coding with SQL. Typically, SQL commands and keywords are not case-sensitive, which allows you to enter your commands and keywords in either uppercase or lowercase—whatever you prefer. The case may be mixed (both uppercase and lowercase for a single word or statement).

Case sensitivity is, however, a factor when dealing with data in SQL. In most situations, data seems to be stored exclusively in uppercase in a relational database to provide data consistency.

For instance, your data would not be consistent if you arbitrarily entered your data using random case:

```
SMITH

Smith

smith
```

If the last name was stored as smith and you issued a query as follows, no rows would be returned.

```
SELECT *
FROM EMPLOYEE_TBL
WHERE LAST_NAME = 'SMITH';
```

Note

You must use the same case in your query as the data is stored when referencing data in the database. When entering data, consult the rules set forth by your company for the appropriate case to be used. The way data is stored varies widely between organizations

Examples of Simple Queries

This section provides several examples of queries based on the concepts that have been discussed. The hour begins with the simplest query you can issue, and builds upon the initial query progressively. You use the EMPLOYEE_TBL table.

Selecting all records from a table and displaying all columns:

```
SELECT * FROM EMPLOYEE_TBL;
```

Selecting all records from a table and displaying a specified column:

```
SELECT EMP_ID
FROM EMPLOYEE_TBL;
```

Selecting all records from a table and displaying a specified column. You can enter code on one line or use a carriage return as desired:

```
SELECT EMP_ID FROM EMPLOYEE_TBL;
```

Selecting all records from a table and displaying multiple columns separated by commas:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL;
```

Displaying data for a given condition:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL
WHERE EMP_ID = '333333333';
```

Displaying data for a given condition and sorting the output:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL
WHERE CITY = 'INDIANAPOLIS'
ORDER BY EMP_ID;
```

Displaying data for a given condition and sorting the output on multiple columns, one column sorted in reverse order:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL
WHERE CITY = 'INDIANAPOLIS'
ORDER BY EMP_ID, LAST_NAME DESC;
```

Displaying data for a given condition and sorting the output using an integer in the place of the spelled-out column name:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL
WHERE CITY = 'INDIANAPOLIS'
ORDER BY 1;
```

Displaying data for a given condition and sorting the output by multiple columns using integers, the order of the columns in the sort is different than their corresponding order after the SELECT keyword:

```
SELECT EMP_ID, LAST_NAME
FROM EMPLOYEE_TBL
WHERE CITY = 'INDIANAPOLIS'
ORDER BY 2, 1;
```

Note

When selecting all rows of data from a large table, the results could render a substantial amount of data returned.

Counting the Records in a Table

A simple query can be issued on a table to get a quick count of the number of records in the table or on the number of values for a column in the table. A count is accomplished by the function COUNT. Although functions are not discussed until later in this book, this function should be introduced here because it is often a part of one of the simplest queries that you can create.

The syntax of the COUNT function is as follows:


```
SELECT COUNT(*)
FROM TABLE_NAME;
```

The COUNT function is used with parentheses, which are used to enclose the target column to count or the asterisk to count all rows of data in the table.

Counting the number of records in the PRODUCTS_TBL table:


```
SELECT COUNT(*) FROM PRODUCTS_TBL;
```


```
COUNT(*)
----------
         9
```

```
1 row selected.
```

Counting the number of values for PROD_ID in the PRODUCTS_TBL table:


```
SELECT COUNT(PROD_ID) FROM PRODUCTS_TBL;
```


```
COUNT(PROD_ID)
--------------
             9

1 row selected.
```

Note

Interesting note: Counting the number of values for a column is the same as counting the number of records in a table, if the column being counted is NOT NULL (a required column). However, COUNT(*) is typically used for counting the number of rows for a table.

Column Aliases

Column aliases are used to rename a table's columns for the purpose of a particular query. The PRODUCTS_TBL illustrates the use of column aliases.

```
SELECT COLUMN_NAME ALIAS_NAME
FROM TABLE_NAME;
```

The following example displays the product description twice, giving the second column an alias named PRODUCT. Notice the column headers in the output.


```
SELECT PROD_DESC,

PROD_DESC PRODUCT

FROM PRODUCTS_TBL;
```


```
PROD_DESC                  PRODUCT
-------------------------- ------------------------
WITCHES COSTUME            WITCHES COSTUME
PLASTIC PUMPKIN 18 INCH    PLASTIC PUMPKIN 18 INCH
FALSE PARAFFIN TEETH       FALSE PARAFFIN TEETH
```

```
LIGHTED LANTERNS          LIGHTED LANTERNS
ASSORTED COSTUMES         ASSORTED COSTUMES
CANDY CORN                CANDY CORN
PUMPKIN CANDY             PUMPKIN CANDY
PLASTIC SPIDERS           PLASTIC SPIDERS
ASSORTED MASKS            ASSORTED MASKS
KEY CHAIN                 KEY CHAIN
OAK BOOKSHELF             OAK BOOKSHELF


11 rows selected.
```

Column aliases can be used to customize names for column headers, and can also be used to reference a column with a shorter name in some SQL implementations.

Note

When a column is renamed in a SELECT statement, the name is not a permanent change. The change is only for that particular SELECT statement.