SIT103 Database and Information Retrieval

Practical 3: SQL Data Manipulation Commands

Objectives:

- To learn more about the SELECT command
- To get familiar with the basic structure of SQL commands

---

1. Create Sample Tables Using a Load Script

A table is used to store data in a database, and a database usually contains many tables. To create sample tables with sample data in your database for practice in the following practical sessions, please do the followings:

- Download the load script file, ***Prac03_Sample.sql***, from CloudDeakin, and save it on your H: drive on Deakin server. If you are now using your home computer, after downloading the file to your computer, use the WINSCP software to upload the file to your H: drive on Deakin server (more details of uploading a file to your H:drive can be found in Practical 2 document).
- Open the script file, ***Prac03_Sample.sql***, using Notepad++ or any another text editor, have a look at the SQL script in the file to understand table structures to be built, how to create a table structure, and how to insert data into tables.
- Connect to your Deakin Oracle database, at the SQL> prompt, execute the following command to create sample tables and insert data into these sample tables:

```
SQL> @Prac03_Sample.sql
```

- Use the SQL commands learnt from Practical 2 to check whether these sample tables are in your database, view the structure and sample data of each table.

2. Introduction

SQL (Structured Query Language) is a database-specific language, i.e., you have to use SQL commands if you are going to communicate with a database through a database management system (DBMS) such as Oracle. The SELECT command is the only SQL query command for retrieving data/information from a database. In this practical, we are going to learn various applications of the command SELECT for data and information retrieval. It will help you understand how to communicate with your database, and get familiar with SQL. You will see that SQL commands are easy to understand.

One of the very first things that you would usually want to do with a set of tables or a database is to see what data the tables contain. To display the data in a table, we use a **SELECT** command on the table.

SELECT is usually the first word in a SQL statement or query. The SELECT statement returns information from a table (or a set of tables, the database) as a set of records, or a result set. The result set is a tabular arrangement of data, composed of rows and columns. The SELECT statement shows the output on the computer screen. It does not save the results. The simplest and most commonly used form of the SELECT syntax is:

```
SELECT fields (a.k.a. columns or attributes)
FROM Table
```

Here, Table is the name of the table from which the data will be retrieved, and fields are the columns (attributes) that you chose to display from the named table. If you did not know the name of the columns in the table, or you wanted to display all the columns in the table, you would use an asterisk (*) in place of fields.

But before we use the SELECT statement, we have to make sure that the right database is open (e.g., log in your Oracle database). Now, your database should have some sample tables created using the loading script. Type the following command at the SQL> prompt, execute it and see what is displayed on the screen:

```
SELECT *
FROM Dependent;
```

In the following sections, by default all SQL commands will be typed and executed at the prompt SQL>. We will learn more about the SELECT command.

3. Displaying or Selecting One, More and All Columns from a Table

Using a SELECT statement, you can display or return one, more than one and all columns from a table. However, to be able to display or return particular fields or columns from a table, you need to know the column names in the table. To view the column names that a table contains, you can use the DESCRIBE (or DESC) command introduced in Practical 2.

Now try the following SELECT statements one by one and see what is displayed on the screen:

```
SELECT dname
FROM Dependent;

SELECT sname
FROM Student;

SELECT dname, relationship
FROM Dependent;

SELECT *
FROM Dependent;
```

Things To Do: Try yourself to display one, more than one and all columns of other tables in your database.

4. ORDER BY

A table maintains the data in the order that the system stores it in, which is unpredictable. Remember that a relational database contains sets of rows of data and sets are not ordered. If you wish to display the contents of a table in a predictable manner, you may use the ORDER BY clause in the SELECT statement.

Try the following statement:

```
SELECT dname, age
FROM Dependent
ORDER BY age;
```

This statement produces the rows of results ordered by age.

The ORDER BY does not actually change the order of the data in the table. It only displays or returns the data output in a particular order.

When using an ORDER BY in a SELECT statement, you do not have to have the column that you are ordering by in the SELECT statement. However, it is generally better to display the column that you are ordering by in the SELECT statement to make the results more understandable.

Try the following statement one by one to see what the difference is. Which one is better?

```
SELECT dname, age
FROM Dependent
ORDER BY sex;
```

```
SELECT dname, age, sex
FROM Dependent
ORDER BY sex;
```

ORDER BY and NULLs

When data has not been entered for a particular column of a particular row, this cell gets a NULL value. NULL means that data is missing or unavailable, so the cell has no value.

If the field that you choose to ORDER BY contains nulls, the fields that have null values assigned to them are placed at the top of the displayed list of output. Try again the following statement and notice the rows with null values:

```
SELECT dname, age
FROM Dependent
ORDER BY age;
```

If nothing was entered in a column (an empty string was entered), the column behaves just like a NULL field when using ORDER BY clause. Try the following statement and notice the rows with nothing in the relationship field:

```
SELECT dname, relationship
FROM Dependent
ORDER BY relationship;
```

Ascending and Descending Order
By default, the order of an ORDER BY is ascending. To display or order output in descending order, the keyword DESC has to be appended to the ORDER BY clause, And, in order to display or order output in ascending order, the keyword ASC can be appended to the ORDER BY clause.

Try the following statement and check the values in age field:

```
SELECT dname, age
FROM Dependent
ORDER BY age DESC;
```

Ordering Within an Order
There will be times when you will want to sort groups within an order by another order. SQL syntax allows you to do this. Try the following statement and check the results:

```
SELECT dname, sex, age
FROM Dependent
ORDER BY sex, age DESC;
```

This statement orders all the dependents by sex, and within sex the dependents are ordered by age in descending order.

Try the following statement and notice the difference:

```
SELECT dname, sex, age
FROM Dependent
ORDER BY sex DESC, age DESC;
```

5. Displaying or Selecting Rows or Tuples from a Table

In relational database terminology, a table is called a *relation*, and is denoted by the name of the relation followed by the columns (or attributes) as shown below:

```
Dependent (pno, dname, relationship, sex, age)
```

An instance of a relation is a row of a relation (table) with values. We will use the term “row” to refer to a line of output, although database literature also uses the term “tuple” or “record” in place of row.

In the previous sections, we showed how to select or display particular columns from a table, but we did not show how to select or display specific rows. Usually you would want to select or display only particular rows from a table.

By using a WHERE clause in a SELECT statement, you can selectively choose rows that you wish to display based on a criterion. For additional filtering, the WHERE clause can be used with logical operators like AND and OR, and the BETWEEN operator and its negation, NOT BETWEEN.

Filtering with WHERE

The WHERE clause is a row filter that is used to restrict the output of rows (or tuples) in a result set. When the WHERE clause is used, the SQL Serve database engine selects rows from the table for the result set that meet the conditions listed in the WHERE clause. If no WHERE clause is used in a query as we did before, the query will return all rows from the table. The following is the general syntax of a SELECT statement with a WHERE clause:

```
SELECT column-names
FROM Table
WHERE criteria
```

Try the following statement:

```
SELECT *
FROM Dependent
WHERE sex = 'F';
```

This query produces rows where the sex attribute has been assigned a value of F.

The WHERE clause can be used with several comparison operators:

>	(greater than)
<>	(not equal)
=	(equal)
>=	(greater than or equal to)
<=	(less than or equal to)

WHERE may be used in a query in addition to ORDER BY. Try the following statement and check the results:

```
SELECT dname, age
FROM Dependent
WHERE age <= 5
ORDER BY age;
```

The AND Operator

The AND operator is a way of combining conditions in a WHERE clause. An AND operator is used in a WHERE clause if more than one condition is required to be met at the same time. Try the following statement and check the results:

```
SELECT *
FROM Dependent
WHERE age <= 5
AND sex = 'F';
```

The OR Operator

The OR operator is another way of combining conditions in a WHERE clause. Unlike the AND operator, the OR operator allows the database engine to select the row to be included in the result set if *either* of the conditions in the WHERE clause are met. Try the following query and check the results:

```
SELECT *
FROM Dependent
```

```
WHERE age > 20
OR sex = 'F';
```

The BETWEEN Operator

The BETWEEN operator is yet another way of combining filtering conditions in a WHERE clause. The BETWEEN operator allows you to determine whether a value falls within a given range of values (inclusive). The general syntax of the BETWEEN operator is:

```
SELECT
FROM
WHERE
BETWEEN value1 AND value2
```

In the BETWEEN clause, `value1` has to be less than `value2`.

Try the following query and check the results:

```
SELECT dname, age
FROM Dependent
WHERE age
BETWEEN 3 AND 5;
```

You can also try the following query to produce the same results:

```
SELECT dname, age
FROM Dependent
WHERE age >= 3
AND age <= 5;
```

Negating the BETWEEN Operator

The BETWEEN operator can be negated by using the keyword NOT before the BETWEEN operator. NOT BETWEEN allows you to determine whether a value does not occur within a given range of values. The general syntax of the NOT BETWEEN is:

```
SELECT
FROM
WHERE
NOT BETWEEN value1 AND value2
```

Try the following query and check the results:

```
SELECT dname, age
FROM Dependent
WHERE age
NOT BETWEEN 3 AND 5;
```

You can also try the following query to produce the same results:

```
SELECT dname, age
FROM Dependent
WHERE age < 3
OR age > 5;
```

IS NULL
The IS NULL condition is the only condition that directly tests for nulls. Rows with null values cannot be retrieved by using = NULL in a WHERE clause, because NULL signifies a missing value. Try the following query and check the results:

```
SELECT dname, age
FROM Dependent
WHERE age IS NULL;
```

IS NOT NULL
To retrieve all rows that are not nulls, IS NOT NULL can be used. Try the following query and check the results:

```
SELECT dname, age
FROM Dependent
WHERE age IS NOT NULL;
```

6. The COUNT Function

The COUNT function is used to return a count of the number of rows that the output will produce, without actually displaying all of the output (rows) themselves. Try the following queries one by one and check the results:

```
SELECT *
FROM Dependent;
```

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

It is often useful to count the occurrence of column values that have a value. Try the following query:

```
SELECT COUNT(age)
FROM Dependent;
```

Here, `COUNT(age)` counts only the rows in which age is not null.

7. Using Aliases

Column aliases and table aliases are temporary names assigned within a query to columns and tables respectively. They are created on the fly in a query, and do not exist after the query is run.

Column Aliases
Column aliases are used to improve the readability of a query and its output. A column alias can be declared after the column designation in the SELECT statement.

Try the following statement:

```
SELECT dname, age, sex
FROM Dependent
WHERE age > 5;
```

In the returned results, notice that Oracle (by default) uses the column names from the Dependent table for the column headings, which are sometime unreadable or not meaningful.

To use more descriptive column headings, you can include column aliases just before or after the column name by using **AS** in the SELECT statement. Try the following statement and notice the column headings:

```
SELECT dname AS Dependent_name, age AS Dependent_age, sex AS
       Dependent_sex
FROM Dependent
WHERE age > 5;
```

The output has more descriptive headings.

You can use only a one-word alias after column name without using **AS** (but a space between the column name and the alias is necessary). Try the following command:

```
SELECT dname Dependent_name, age Dependent_age,
        sex Dependent_sex
FROM Dependent
WHERE age > 5;
```

To embed a blank in the column alias, you have to put the column alias in single or double quotes. Try the following:

```
SELECT dname AS "Dependent Name", age As "Dependent Age",
       sex AS "Dependent Sex"
FROM Dependent
WHERE age > 5;
```

Table Aliases

A table alias, usually used in multi-table queries (we will discuss multi-table queries in later sessions), allows us to use a shorter name for a table when we reference a table in the query. A table alias is temporary, and does not exist after the query is run. Try the following statement:

```
SELECT d.dname
FROM Dependent d
WHERE d.age > 5;
```

In this query, the table alias is the letter **d** *after* the table name `Dependent`. A table alias can also be defined by a short, meaningful word or expression after the table name, rather than a one-letter table alias, but the one-letter table alias is commonly used by SQL programmers.

Again note that the table alias is not valid outside this query. That is, if you try `SELECT * FROM d`, you will get an error message because there is no such table as d.

In the above example, the construction **`d.dname`** contains a table qualifier (the **`d.`** part). **Table qualifiers are needed when the same column name has been used in more than one table**. Table qualifiers before the column names determine which table the column is from. For example, if `TableA` has a column called `Field1` and

`TableB` also has a column `Field1`, if we do not use a table qualifier in a multi-table query, there is no way that the query engine can know which `Field1` the query is referring to. To correctly handle this situation, you would have to use a table qualifier in the form `TableA.Field1`, where `TableA` is the table qualifier (this is also an alias in a way).

The advantage of using table qualifiers and table aliases may not be so apparent in the examples presented here, because we are now working only with single table. As we start working with multiple tables later, their advantages will become more obvious.

8. Some Conventions for Writing SQL Statements

Although SQL statements often contain multiple commands and multiple lines, there are no fixed rules for writing SQL statements; SQL is a "free-form" language. We suggest that you use the following conventions to increase the readability of your queries, especially as your statements or queries become more complex:

- User uppercase letters for the keywords, which includes SELECT, FROM, and WHERE. Use lowercase letters for the user-supplied words (Oracle is not case sensitive for commands).
- Align the keywords SELECT, FROM, and WHERE on separate lines, like the examples above.

9. Summary

In this practical session, you have learnt

- How to use the basic SELECT statement and how to extract columns and rows using SELECT;
- How to use COUNT function, the AND, OR, and BETWEEN operators, table and column aliases;
- The concept of nulls
- Some conventions for writing SQL statements.

You will need this basic knowledge and understanding to work the forthcoming sessions.

10. Things To Do

Practise the SQL commands introduced in this session on other tables in your database.
