SIT103 Database and Information Retrieval

Practical 2: More SQL*Plus and SQL Commands in Oracle

Objectives:

- To learn more SQL*Plus and SQL commands for table and data manipulation in a database
- To extend your existing SQL command knowledge via examples and exercises

---

Introduction:

In Practical 1, we learned

- how to connect to an Oracle databases,
- how to create tables in a database and populate tables with data by using SQL commands CREATE TABLE and INSERT INTO,
- how to use SQL*Plus command DESCRIBE (or DESC) to view a table structure,
- how to use SQL command SELECT to view the created tables in a database and the data in a table, and
- how to exit the database.

Generally speaking, at the SQL> prompt we can execute standard SQL commands. In addition, Oracle also provides some SQL*Plus commands to manipulate objects in a database. Getting to know these specific commands will help us use the database system effectively and efficiently.

In this practical, we introduce more SQL*Plus and SQL commands in Oracle.

1. Batch execution of SQL commands

In practical 1, you might have noticed that SQL*Plus command environment or screen only provides a line editor for typing in and executing a SQL command. The user does not have control over the screen. The line editor does not allow the user to move the cursor up and down, and clicking with a mouse is definitely out of the question.

To avoid these problems, you can use alternative ways to edit and execute SQL commands in a batch. Two recommended ways are presented below.

i) Using another text editor.

You can use another text editor, such as **Notepad++**, to type your SQL commands. Within the text editor, you can do full-screen editing. The SQL command typed in a full-screen text editor can be copied to the clipboard, and then pasted into SQL*Plus screen to execute it at the SQL> prompt.

EXERCISE: Open the text editor Notepad++, type in the following SQL command within the Notepad++:

```
SELECT employeeName
FROM works
WHERE salary > 15000;
```

Make changes to the typos. Copy this command, paste it at the SQL> prompt, and then execute it.

ii) Edit and execute a SQL command file.

If you are now using a computer in a Deakin lab, you can use a text editor, such as Notepad++, to type your SQL commands and then save them in a file with a ***.sql*** extension **on your H: drive**. At the SQL> prompt, type a command in the format *@filename.sql* to execute the SQL commands in the saved ***.sql*** file:

EXERCISE: Open the text editor Notepad++, type in the following SQL commands within the Notepad++:

```
CREATE TABLE staff
(staffName CHAR(15) NOT NULL,
 street    CHAR(15),
 city      CHAR(10),
 PRIMARY KEY (staffName));

INSERT INTO staff
VALUES ('David', 'Huntingdale', 'Whitehorse');

INSERT INTO staff
VALUES ('Harry', 'Main', 'Kew');

INSERT INTO staff
VALUES ('Mary', 'Cliffton', 'Monash');
```

Save these commands in the file ***ex1.sql*** on your **H: drive**, and then go back to your SQL*Plus environment (screen) to execute it:

```
SQL> @ex1.sql
```

Check your database to see whether table *staff* exists:

```
SQL> SELECT table_name FROM user_tables
   2 WHERE table_name = 'STAFF';
```

Display the data in table *staff*:

```
SQL> SELECT * FROM staff;
```

[**NOTE:** It can be seen that you can type in many SQL commands in a *.sql* file, and execute these command in a batch, instead of executing them one by one by yourself.]

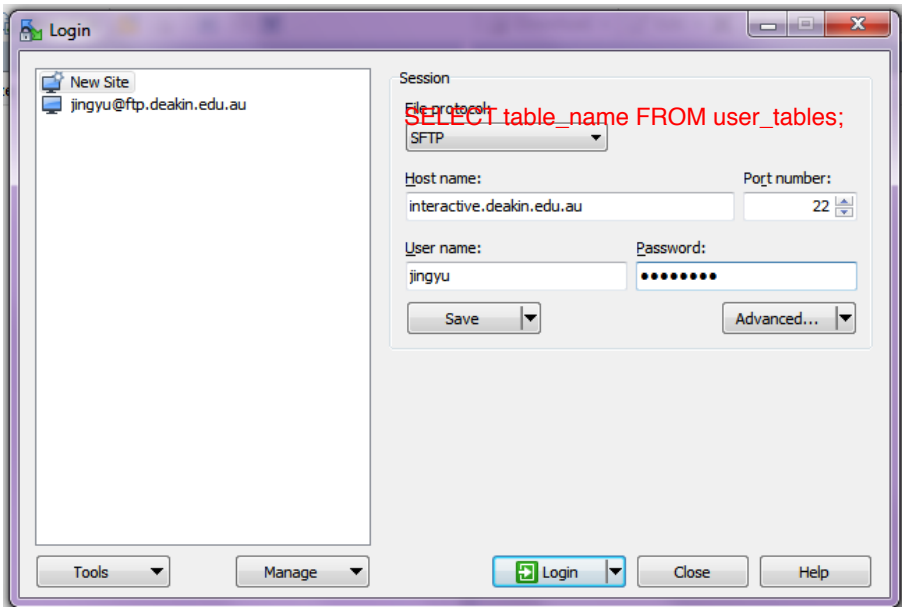
If you are now using your **home computer**, you can also use a text editor, such as Notepad++, to type your SQL commands, save them in a file with a

.sql extension on your home computer. Then **upload the *.sql* file to your H: drive on the Deakin server** (interactive.deakin.edu.au). The execution of the ***.sql*** file is the same as above.

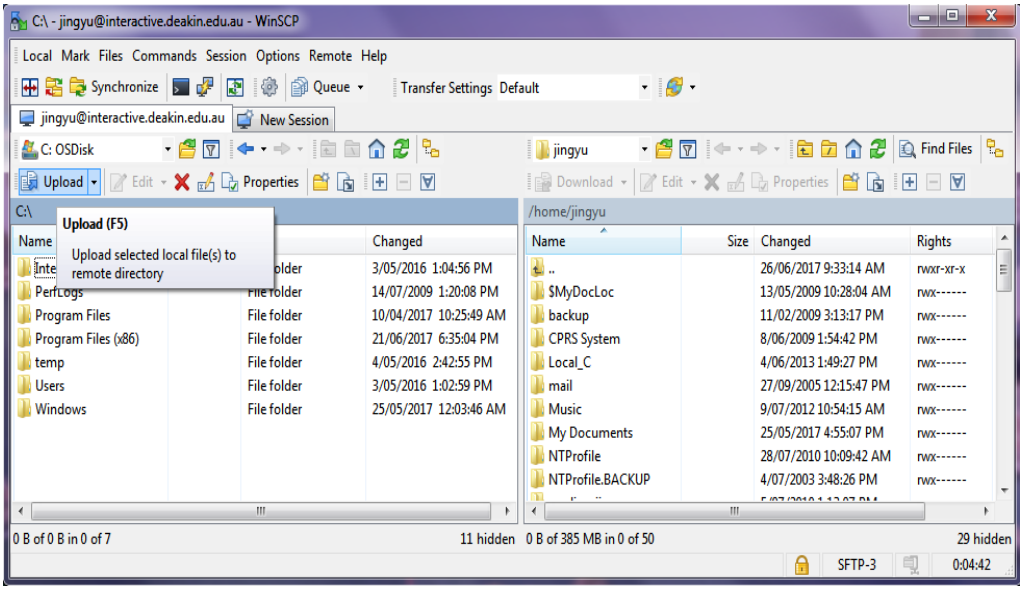
You can use any FTP software, such as WinSCP, to upload your ***.sql*** file to your H: drive on the Deakin server. The WinSCP software can be downloaded from the following Deakin software library website:

http://software.deakin.edu.au/2017/03/24/winscp-windows-secure-copy/

After installing and launching the WinSCP, connect to the Deakin server (*interactive.deakin.edu.au*) by selecting "New Site" and entering your Deakin username and password like this:



Click "Login" button to connect to the server (*interactive.deakin.edu.au*). When the connection is successful, the WinSCP interface looks like this (yours might be a little bit different):



The left panel is your local computer, while the right panel is the remote host (i.e., your H: drive on Deakin server).

- Select the ***.sql*** file you are going to upload from your local computer (left panel).
- Select the directory on the remote host (right panel) where you are going to save the uploaded file. For the above exercise, you don't need to select another directory, as you are now on your H: drive.
- Click the "Upload" button as shown in the above screen-shot to upload the file to the remote host (i.e., your H: drive).

  NOTE: The username (e.g., "*jingyu*") showing on the top of right panel means your home directory on Deakin server (i.e., H: drive).

Alternative Way of Accessing H: drive on Deakin Server:

You can also map your H: drive on Deakin server to a local drive (e.g., Z: drive) on your own/local computer. So you can directly save your SQL files on H: drive from your local computer.

To do this you must first establish a secure (VPN) connection to Deakin using Pulse Secure, then follow the instructions to map your home directory (Windows), for Mac users follow the instructions (Mac OS X).

EXERCISE: Do the above exercise by saving the commands in the file ***ex1.sql*** on your home computer, upload/save it to your **H: drive** on Deakin server, and then execute the file as the above.

2. Help Facilities

At the SQL > prompt, type `help` followed by a keyword or sequence of keywords that are used together in SQL commands, for example,

```
SQL> help select
```

or

```
SQL> help create table
```

If you are lucky, the keyword or phrase will be among those for which help exists, and you will get a (somewhat) helpful message, usually ending in an example or two.

Sometimes, the results of a SQL command execution may be too long to display on a screen. If so, you can issue

```
SQL > set pause on
```

to activate the paging feature. When this feature is activated, output will pause at the end of each screen until you hit the "Return" key. You can try the above commands again under this setting to see the effect.

To turn this feature off, execute

```
SQL > set pause off
```

3. Table Management Commands

RENAME Command

The RENAME command is used to change the name of the database object, such as tables and views. To RENAME a database object, the syntax is:

```
RENAME old TO new;
```

EXERCISE: Please try to rename table *employee* to *worker*, list the data in table *worker*, and then rename *worker* back to *employee*.

Add a Column and / or Constraints

You can use ALTER TABLE … ADD command to add a new column and / or constraints to an existing table. The syntax is

```
ALTER TABLE <tableName>
ADD (column_name [column_constraint]);
```

Try the following commands to see how a new column is added to an existing table:

```
SQL> ALTER TABLE staff
   2 ADD (manager char(10));
…
SQL> DESCRIBE staff;
…
```

Modify Columns

You can use the ALTER TABLE … MODIFY command to modify the definitions of existing columns. The syntax is

```
ALTER TABLE <tableName>
MODIFY (column_name type[,column_name type, …]);
```

Try to modify the width of the *manager* column in table *staff* to 15 characters, enter:

```
SQL> ALTER TABLE staff
   2 MODIFY (manager char(15));
…
SQL> DESCRIBE staff;
…
```

Please note, there are four changes that you can not make:

a) You may not change a column containing nulls from NULL to NOT NULL.

b) You may not add a new column that is NOT NULL. Make it null, fill it completely and then change it to NOT NULL.

c) You may not decrease the size of a column or change its data type, unless it contains no data.

d) You may not use the MODIFY option to define constraints on a column except for NULL/NOT NULL.

Delete a Column

To delete a column from a table, you can use the ALTER TABLE … DROP command. The syntax is

```
ALTER TABLE <tableName>
DROP COLUMN <columnname>
```

EXERCISE: Try to do the followings:

- Add a column *e_mail* to table *staff*, the data type is CHAR(25);
- View the structure of table *staff*;
- Delete the column *e_mail* from table *staff*;
- View the structure of table *staff*.

Delete a Table

To delete or remove a created table from your database, use the DROP command in the following format:

```
SQL > DROP TABLE <table_name>;
```

For example, if there is a table *test* in your database and you want to delete it, type and execute the following command:

```
SQL > DROP TABLE test;
```

If you don't want some tables to stay in your database, you can remove them at any time you prefer. This will avoid leaving a lot of garbage around that will be still there the next time you use the Oracle system.

4. Data Manipulation Commands

We have learnt some data manipulation commands in the previous practical, such as `insert` command for inserting rows into a table. Here, we learn more data manipulation commands.

Copy Rows from another Table

You can use the following `INSERT` statement format to insert/copy several rows into a table where the values are derived from the contents of existing tables in the database.

```
INSERT INTO tableName[(column, column, …)]
       SELECT select_list
       FROM table(s)
       [WHERE condition(s)];
```

[**Note**: the keyword "VALUES" is not used in this command format.]

EXERCISE: We have already created a new table *staff* from the above exercise. The structure of table *staff* is the same as table *employee* except the last column *manager* in table *staff*. Try to do the followings:

- Insert the employee data into table *staff* table using the following command:

```
SQL> INSERT INTO staff (staffName, street, city)
     2 SELECT employeeName, street, city
     3 FROM employee;
```

- Check the data in table *staff* to see if it is correct.

Update Rows

The UPDATE statement allows you to change values in rows in a table. The syntax is as follows:

```
UPDATE <tableName>
SET column [, column, …] = {expression, subquery}
[WHERE condition];
```

Try the following command to increase the salary of Jones by 10%, and check the updated result:

```
SQL> UPDATE works
2 SET salary = salary * 1.1
3 WHERE employeeName = 'Jones';
…
SQL> SELECT * FROM works WHERE employeeName = 'Jones';
…
```

If the WHERE clause is omitted, all salaries in the table will be increased by 10%. Try the following commands:

```
SQL> UPDATE works
   2 SET salary = salary * 1.1
   …
SQL> SELECT * FROM works;
…
```

EXERCISE: Now that we have a new table *staff*, please insert **manager** data for each staff in the *staff* table. (Hint: use the UPDATE command.)

Delete Rows from a Table

The DELETE command allows you to remove one or more rows from a table. The syntax is as follows:

```
DELETE FROM <tableName>
[WHERE condition];
```

If the WHERE clause is omitted, *all rows* will be deleted.

Another way to delete all rows from a table is to use the command TRUNCATE. The syntax is as follows:

```
TRUNCATE TABLE <tablename>;
```

EXERCISE: Try to insert a row into a table, and then delete this new row from this table. Then delete all rows from a table.

5. Substitution Variables

You might have noticed in the previous exercises that many SQL commands are re-typed with various different values. For example, when you try to insert rows/tuples into a table, you have to re-type the `insert` command many times. This wastes much time in tedious work. Using substitution variables, you can simplify and speed up your work.

Substitution variables are used to store elements of command text, which is "edited" into commands before their execution.

- Single Ampersand Substitution Variables

  You can use substitution variables in a command file or single SQL statement to represent values to be provided at runtime. A variable can be thought of as a container in which values are stored temporarily.

  A substitution variable is prefixed by a single ampersand (&), and a value is assigned to it. For example, the following statement prompts the user to provide actual values for `employeeName`, `street` and `city` in table *employee*:

```
SQL> INSERT INTO employee
   2 VALUES('&Name','&Street','&City');
```

  The second or next time you want to insert another row/tuple into table *employee*, you just need to re-run this command by typing RUN (or /). The system will prompt you to provide new values. Please try the above command to insert several rows into table *employee*. You can also try to use this method to insert many rows into other tables.

  In the above example, the substitution variable for *character* or *date* value is enclosed in single quotes, thus at run time, the user need not enclose the actual values in single quotes. Otherwise, the actual values must be enclosed in single quotes. For other type value, you can use the substitution variables directly. Try the following command:

```
SQL> INSERT INTO works
   2 VALUES('&Name','&CoName', &Salary);
```

  Since the data in a substitution variable is inserted into a command before execution, the variable may be used to provide run-time values to any part of a command's structure except for the command name itself. Try the following commands:

```
SQL> SELECT employeeName, &arithmetic_expression
   2 FROM works;
Enter value for arithmetic_expression: salary/12
```
…

```
SQL> SELECT * FROM works
   2 WHERE &Condition;
Enter value for Condition: salary > 20000
```
…

```
SQL> SELECT &query;
Enter value for query: (input the query expression you would like)
```
…

With the single ampersand substitution variable, the user is prompted every time the command is executed, because the variable is not defined and consequently the value entered is not saved. Also as many substitution variables can be used as you like, provided they are preceded by an &.

Concatenation can also occur with substitution variable. A dot (.) acts as the concatenation character, for example:

```
SQL> SELECT * FROM works WHERE salary =&x.000;
Enter value for x: 21
old 1 : select * from works where salary =&x.000
new 1 : select * from works where salary =21000

EMPLOYEENAME     COMPANYNAME           SALARY
------------     --------------------  ------------
Jones            Tweeties              21000
```

- Double Ampersand Substitution Variables

If a variable is prefixed with double ampersand (&&), SQL*Plus will only prompt for the value once. SQL*Plus stores the first value supplied and uses it again whenever the SQL statement is run. Try the following command ***more than two*** times to see how the double ampersand variable works:

```
SQL> SELECT employeeName, salary
   2 FROM works
   3 WHERE &&condition;
```

You can use the SQL*PLus `DEFINE` or `DEF` command to check whether a variable is already defined. If the variable is defined, it displays the assigned value. Try the command:

```
SQL> define
```

Variables may also be "emptied" by using the `UNDEFINE` or `UNDEF` command. For example:

```
SQL> undefine condition
```

6. Arithmetic Expression

Arithmetic expression can be used in a query command to enhance the query function. Arithmetic expression may contain column names, constant numeric values and the arithmetic operators. For example, try the following command to display the annual income of each employee when the salary is increased by $250:

```
SQL> SELECT employeeName, (salary+250)*12
   2 FROM works;
```

7. Group Functions

Group functions operate on sets of rows. They return results based on groups of rows, rather than one result per row as returned by single row functions. By default all rows in a table are considered as one group.

Commonly used group functions are as follow:

Function	**Value Returned**
AVG (*expre*)	Average value of *expre*, ignoring null values
COUNT (*expre*)	Number of rows where *expre* evaluates to something other than NULL.
MAX (*expre*)	Maximum value of *expre*.
MIN (*expre*)	Minimum value of *expre*.
SUM (*expre*)	Sum values of *expre*, ignoring null values.

For example, to count the number of employees in table *works* with *salary* > *20000*, execute the following SQL command:

```
SQL> SELECT COUNT(employeeName)
   2 FROM works WHERE salary > 20000;
```

To find the average salary of employees who work for **Meyer**, enter:

```
SQL> SELECT AVG(salary)
   2 FROM works WHERE companyName='Meyer';
```

8. Summary

- Execute SQL commands in a batch.
- Help facilities in Oracle SQL*Plus.
- Table management commands.
- Data manipulation commands.
- Substitution variables.
- Arithmetic expressions in SELECT command.
- Group functions.

Up to now, we have learnt some commonly used SQL commands, especially the SELECT command which is the only SQL command for data and information retrieval from a database. Please notice the usage of SELECT command in all examples and exercises, and try to understand how to use it. In the following practicals, we are going to learn more about the SELECT command.