

Ahsanullah University of Science and Technology
Department of Computer Science and Engineering
Course No: CSE 4126
Course Title: Distributed Database Systems Lab
LAB MANUAL – 1

Objective: The main target of today's lab session is to getting started with Oracle 10g and PL/SQL. After the session, the students will be able to –

- Work on SQL*PLUS.
- Execute SQL commands via scripts.
- Create PL/SQL anonymous block structure, declare variable and execute a statements inside the block.

Software: Oracle 10g, SQL*PLUS

Pre-requisite: Relational schema, SQL, C, C++

Reference:

- [1] <http://www.plsqltutorial.com/> (Most of the contents of this manual are taken from here)
 - [2] https://www.w3schools.com/sql/sql_intro.asp (For SQL revision)
-

What is PL\SQL:

PL/SQL is a Procedural Language (PL) that extends the Structured Query Language (SQL).

It is difficult to write applications using SQL only because each SQL statement runs independently and has little or no effect on each other. To overcome this limitation, you often have to use other programming languages (such as C#, PHP, and Java) as frontend. Oracle, an object-relational database management system produced by Oracle Corporation, supports this approach when you want to develop applications that interact with Oracle databases.

History:

Oracle introduced PL/SQL (version 1.0) in its Oracle Database product version 6.0 as the scripting language in SQL*Plus and programming language in SQL*Forms 3.

Since version 7, Oracle did a major upgrade for PL/SQL (version 2.0) that provides more features such as procedures, functions, packages, records, collections, and some package extensions.

Advantages of PL\SQL:

PL/SQL is a highly structured language: PL/SQL provides a very expressive syntax that makes it easy for anyone who wants to learn PL/SQL. If you are programming in other languages, you can get familiar with PL/SQL very quickly and understand the intent of the code without difficulty.

PL/SQL is a portable and standard language for Oracle development: Once you develop a PL/SQL program in an Oracle Database, you can move it to the other Oracle Databases without changes, with the assumption that the versions of Oracle database are compatible.

PL/SQL is an embedded language: PL/SQL programs such as functions and procedures are stored in Oracle database in compiled form. This allows applications or users to share the same functionality stored in Oracle database. PL/SQL also allows you to define triggers that can be invoked automatically in response to particular events in associated tables.

PL/SQL is a high-performance language inside Oracle Databases: Oracle adds many enhancements to the PL/SQL to make it more efficient to interact with Oracle databases.

Installation:

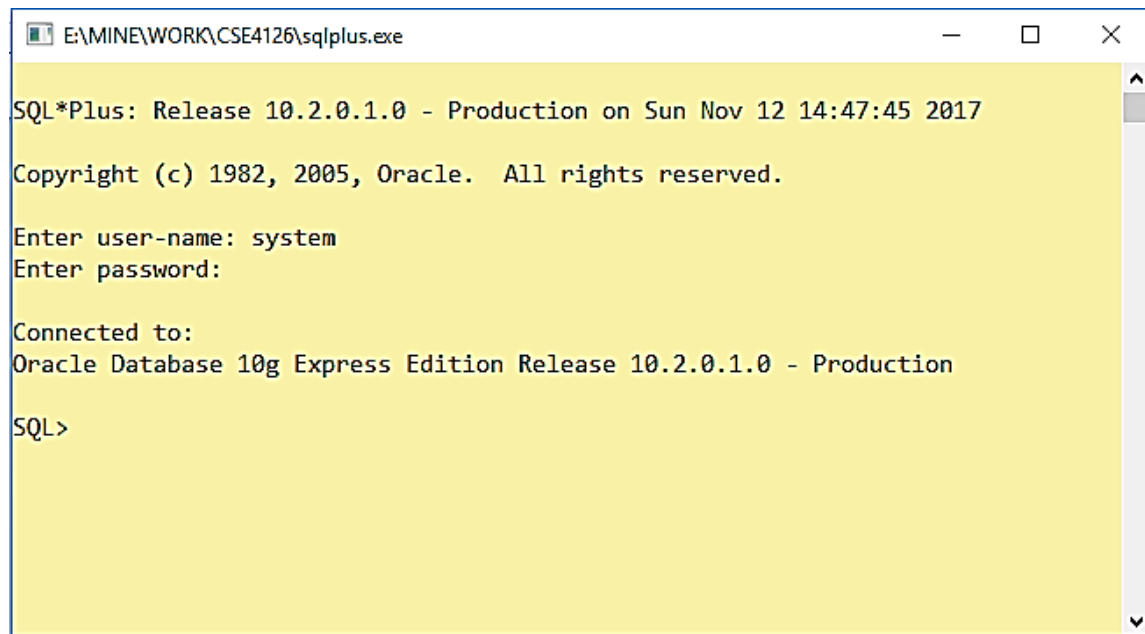
Follow class lectures and provided slides.

Getting started with SQL*Plus:

SQL*Plus is a command line interface tool which you can use to interact with Oracle databases. After installing Oracle 10g, you can access this app at:

C:\oracle\app\oracle\product\10.2.0\server\BIN

After launching it, enter username and password that you have set in the installation process. We enter `system` as user and its corresponding password.

A screenshot of a Windows command window titled "E:\MINE\WORK\CSE4126\sqlplus.exe". The window has a yellow background. The text inside shows the SQL*Plus release information: "SQL*Plus: Release 10.2.0.1.0 - Production on Sun Nov 12 14:47:45 2017". It then displays the copyright notice: "Copyright (c) 1982, 2005, Oracle. All rights reserved." The user is prompted to enter a username and password. The user enters "system" for the username. The password is entered but not displayed. The window then shows the connection status: "Connected to: Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production". The prompt "SQL>" is visible at the bottom.

```
E:\MINE\WORK\CSE4126\sqlplus.exe

SQL*Plus: Release 10.2.0.1.0 - Production on Sun Nov 12 14:47:45 2017

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Enter user-name: system
Enter password:

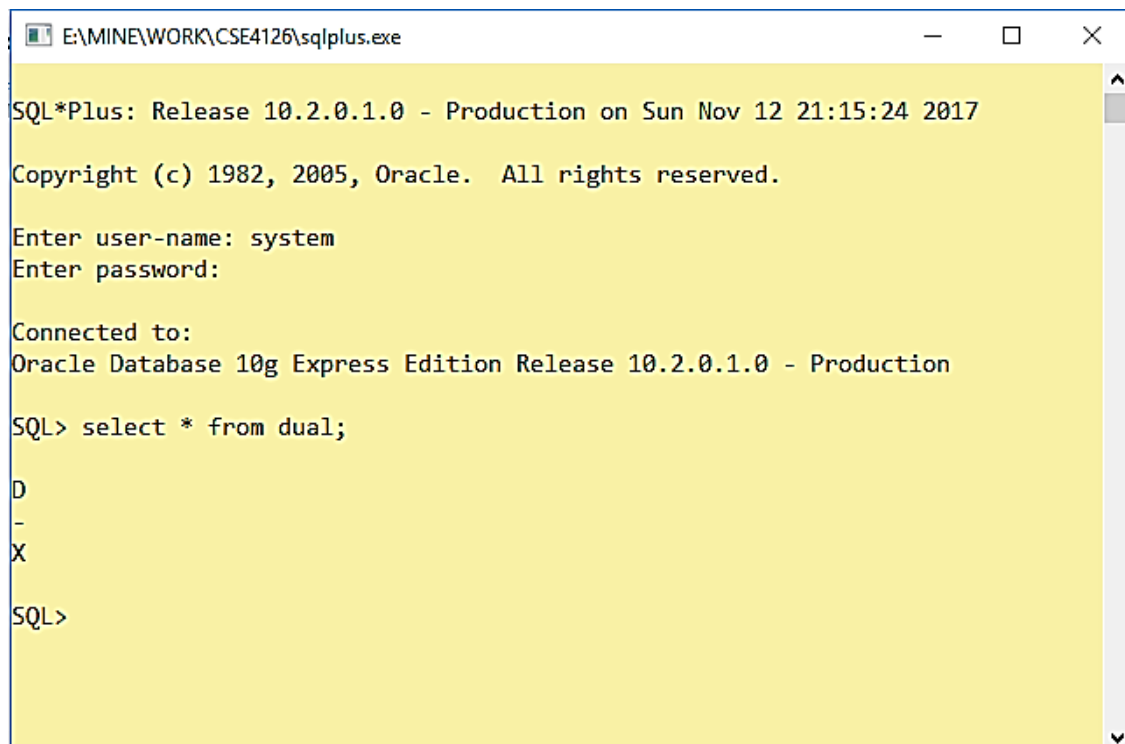
Connected to:
Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production

SQL>
```

Now enter the following command.

```
select * from dual;
```

If you see the output as shown in the following screenshot, it means you have installed Oracle database successfully and start learning PL/SQL programming.

A screenshot of a Windows command window titled "E:\MINE\WORK\CSE4126\sqlplus.exe". The window has a yellow background. The text inside shows the SQL*Plus release information: "SQL*Plus: Release 10.2.0.1.0 - Production on Sun Nov 12 21:15:24 2017". It then displays the copyright notice: "Copyright (c) 1982, 2005, Oracle. All rights reserved." The user is prompted to enter a username and password. The user enters "system" for the username. The password is entered but not displayed. The window then shows the connection status: "Connected to: Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production". The user enters the command "select * from dual;". The output is displayed: "D", "-", "X". The prompt "SQL>" is visible at the bottom.

```
E:\MINE\WORK\CSE4126\sqlplus.exe

SQL*Plus: Release 10.2.0.1.0 - Production on Sun Nov 12 21:15:24 2017

Copyright (c) 1982, 2005, Oracle. All rights reserved.

Enter user-name: system
Enter password:

Connected to:
Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production

SQL> select * from dual;

D
-
X

SQL>
```

Revising Basic SQL commands:

Now let's play around with some DDL and DML of SQL. Note that we will not go through these deeply as they are pre-requisite.

Create table first, then insert one row.

```
CREATE TABLE student (id number(20), name varchar2(20), semester integer, date_of_birth date);
```

```
INSERT INTO student VALUES(1, 'Rahim', 1, '10-oct-1990');
```

```
INSERT INTO student VALUES(1, Karim, 2, '12-oct-1990');
```

The commands are not case-sensitive. It is conventional to have them capitalized. However, we will use small letters in this manual.

Changes made to the database by INSERT, UPDATE and DELETE commands are temporary until explicitly committed. This is performed by the command.

```
commit;
```

To drop a table,

```
drop table student;
```

Let's re-create the student table with primary key and insert data.

```
create table student (id number(20), name varchar2(20), semester integer, date_of_birth date, primary key(id));
```

```
insert into student values(1, 'Rahim', 1, '10-oct-1990');
```

```
insert into student values(2, 'James', 2, '11-jan-1990');
```

```
insert into student values(3, 'Jamal', 3, '13-mar-1990');
```

Create another table called student_result with foreign key id referencing to id of student_table. Here student table is parent and student_result is child.

```
create table student_result (id number(20), cgpa number(6,5), foreign key (id) references student(id));
```

Insert data –

```
insert into student_result values(1, 3.99);
```

```
insert into student_result values(2, 3.85);
```

```
insert into student_result values(3, 2.99);  
  
commit;
```

Try update and delete.

Dropping a table with referential integrity (foreign key) is not straight forward. You need to use following command –

```
drop table student cascade constraints;  
  
drop table student_result cascade constraints;
```

Now working directly on the command prompt can be tedious! We can store our SQL command in a text file/ script and execute that.

We have provided two files `table.sql` and `insert.sql`. First one contains all the commands to create some tables and second one for inserting data into them. Say, we put them in the same directory `E:/MINE/WORK/CSE4126/DB/`.

Now enter the command –

```
@ E:/MINE/WORK/CSE4126/DB/table.sql;  
  
@ E:/MINE/WORK/CSE4126/DB/insert.sql;
```

Also, we can define a variable that contains the directory and thus shorten the command. To do that,

```
define dir = E:/MINE/WORK/CSE4126/DB
```

Now, we can use the following command –

```
@ &dir/table.sql;  
  
@ &dir/insert.sql;
```

If you look inside the `table.sql`, drop commands are given at the beginning. This is to drop any existing table before creating. If there is no table to drop, it may show error message.

Note that, you can provide any name of the scripts and it is not mandatory to have file extension as `.sql`, it could be `.txt` as well.

Comments:

```
/* for multiple lines */
```

```
-- for single line
```

As a practice, create another script named `select.sql` containing selection operation. And from now on, throughout this manual, assume that all commands are executed via scripts.

Now we will revise the following basic SQL commands and topics from relational algebra–

1. Join
2. Sub-query (or nested query)
3. Set operations
4. View

To keep the scripts organized, let's create a folder called `QUERY` where we save all the scripts for query purpose.

1. Join:

Create `join.sql` and write the following commands –

```
select S.name, B.b_group
from student S, student_blood_group B
where S.id = B.id;
```

To run this command –

```
@./QUERY/join.sql;
```

This will output a simple join operation. We can do the same using `join` operator as well.

```
select S.name, B.b_group
from student S inner join student_blood_group B
on S.id = B.id;
```

You can try right join, left join, full join, outer join etc.

2. Sub-query:

For convenience, we put the name of the script as a comment at the top.

```
/* subquery.sql*/

select cgpa from student_result
where id = (select id
from student
```

```
where name = 'Kavin');
```

3. Set:

```
/* set.sql*/  
  
select id from student  
union  
select id from student_contact;
```

Can you think of other set operations?

4. View:

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

```
/* view.sql*/  
  
create or replace view myview as  
  
select S.id, S.name, R.cgpa  
  
from student S, student_result R  
  
where S.id = R.id;  
  
select * from myview;
```

myview will be treated as single table, though it's columns come from student and student_result.

We can set different name of the field of a view than the original table.

```
/* view2.sql*/  
  
create or replace view  
  
myview(view_id, view_cgpa) as  
  
select S.id, R.cgpa  
  
from student S, student_result R  
  
where S.id = R.id;
```

```
select * from myview;
```

What will happen if we modify a value in view? Will it affect the original table?

You definitely should revise other SQL commands, such as –

1. alter
2. aggregate functions, scalar functions
3. having
4. group by
5. like
6. exists, not exists
7. check

Some other topics must be revised, such as – different data types, alias etc. A very good reference to brush up your SQL skill is [Reference no. 2].

PL/SQL Block Structure

PL/SQL program units organize the code into blocks.

A block without a name is known as an anonymous block.

The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.

An anonymous block is an only one-time use and useful in certain situations such as creating test units.

The following illustrates anonymous block syntax:

```
[DECLARE]
    Declaration statements;
BEGIN
    Execution statements;
[EXCEPTION]
    Exception handling statements;
END;
/
```

The anonymous block has three basic sections that are the declaration, execution, and exception handling. Only the execution section is mandatory and the others are optional.

- The declaration section allows you to define data types, structures, and variables. You often declare variables in the declaration section by giving names, data types, and initial values.
- The execution section is required in a block structure and it must have at least one statement. The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section.
- The exception handling section is starting with the EXCEPTION keyword. The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.

Notice that the single forward slash (/) is a signal to instruct SQL*Plus to execute the PL/SQL block.

Let's take a look at the simplest PL/SQL block that does nothing. For convenience, we can create a folder PLSQL and put the codes in null.sql.

```
/* null.sql */
BEGIN
  NULL;
END;
/
```

We save this block in a script say anyonblock.sql. If we execute this, the message will say –

PL/SQL procedure successfully completed.

Now, try this –

```
/* anyonblock.sql */
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello PL/SQL');
END;
/
```

It displays database's output on the screen. Here –

- SET SERVEROUTPUT ON command to instruct SQL*Plus to echo database's output after executing the PL/SQL block.
- The SET SERVEROUTPUT ON is SQL*Plus command, which is not related to PL/SQL.
- The DBMS_OUTPUT.PUT_LINE is a procedure to output a string on the screen.

PL/SQL variables:

Declare:

Before using a variable, you need to declare it first in the declaration section of a PL/SQL block.

Let's see an example –

```
/* variable.sql */  
SET SERVEROUTPUT ON;  
  
DECLARE  
    n_id number;  
    v_name varchar2(20);  
  
BEGIN  
    select id, name  
    into n_id, v_name  
    from student  
    where id = 3;  
  
    DBMS_OUTPUT.PUT_LINE(n_id);  
    DBMS_OUTPUT.PUT_LINE(v_name);  
  
END;  
/
```

Anchors:

PL/SQL provides you with a very useful feature called variable anchors. It refers to the use of the %TYPE keyword to declare a variable with the data type is associated with a column's data type of a particular column in a table.

```
/* variable.sql */  
SET SERVEROUTPUT ON;  
  
DECLARE  
    n_id student.id %TYPE;  
    v_name student.name %TYPE;  
  
BEGIN  
    select id, name  
    into n_id, v_name
```

```

        from student
        where id = 3;

DBMS_OUTPUT.PUT_LINE(n_id);
DBMS_OUTPUT.PUT_LINE(v_name);

END;
/

```

Assignment:

In PL/SQL, to assign a value or a variable to another, you use the assignment operator (:=) . Let's look at the following code for better understanding.

```

/* variable.sql */

SET SERVEROUTPUT ON;

DECLARE
    n_id student.id %TYPE;
    v_name student.name %TYPE;
    v_newname student.name %TYPE;

BEGIN
    v_newname := 'Mamun';

    update student set name = v_newname
    where id = 3;

    select id, name
    into n_id, v_name
    from student
    where id = 3;

    DBMS_OUTPUT.PUT_LINE(n_id);
    DBMS_OUTPUT.PUT_LINE(v_name);

END;
/

```

Initialization:

You can initialize variable a value in declaration section by using variable assignment.

```

SET SERVEROUTPUT ON;

```

```
DECLARE
    n_id student.id %TYPE;
    v_name student.name %TYPE;
    v_newname student.name %TYPE := 'Sohel';

BEGIN
    update student set name = v_newname
    where id = 3;

    select id, name
    into n_id, v_name
    from student
    where id = 3;

    DBMS_OUTPUT.PUT_LINE(n_id);
    DBMS_OUTPUT.PUT_LINE(v_name);

END;
/
```

[END]

- Prepared by
Mohammad Imrul Jubair