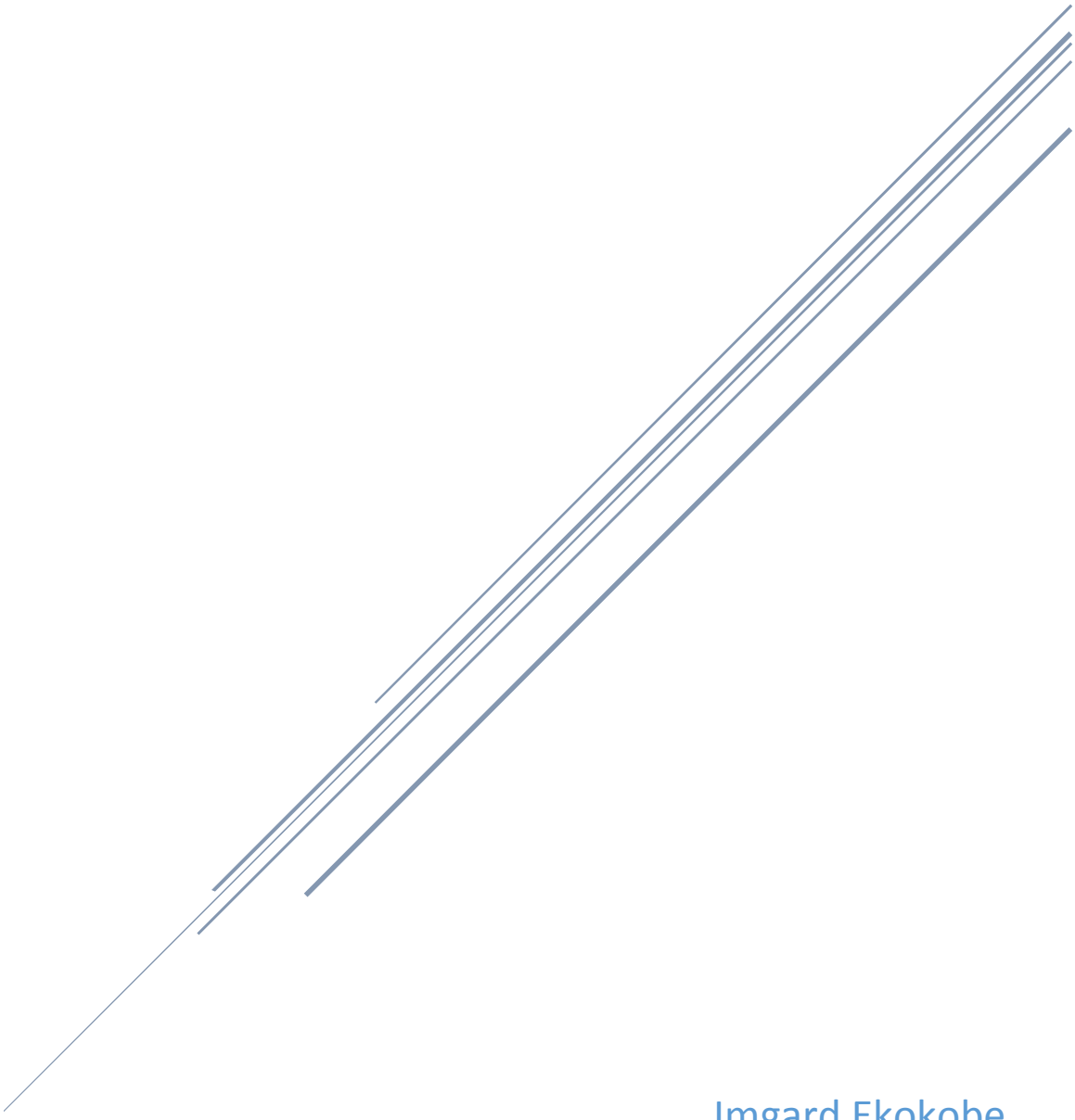
INNOVATIVE DBA

Hands On Practice Workbook



Imgard Ekokobe
Kudrant Academy

Table of Contents

READ ME FIRST READ ME FIRST

Introducing you to all the important basic create table concepts

The best way to learn how to create database tables for a small database is to learn the syntax and then practice and practice how to create them.

These beginning exercises will help you learn how to create simple and complex tables.

The goal of the exercises in this workbook is to give you a feeling for the basic environment typically found in which databases are developed.

I will walk you through creating a simple and complex table by capturing and pasting the commands in this workbook and by using screenshots to support your understanding of the concepts.

Study and review the examples and practice along until you are comfortable with the SQL statements used to create, load and retrieve the tables.

After that, you will ready to complete the assignment at the end of the workbook.

Following directions is KEY in this project. Pay attention to the requirements for the lab assignment.

Keep in mind!

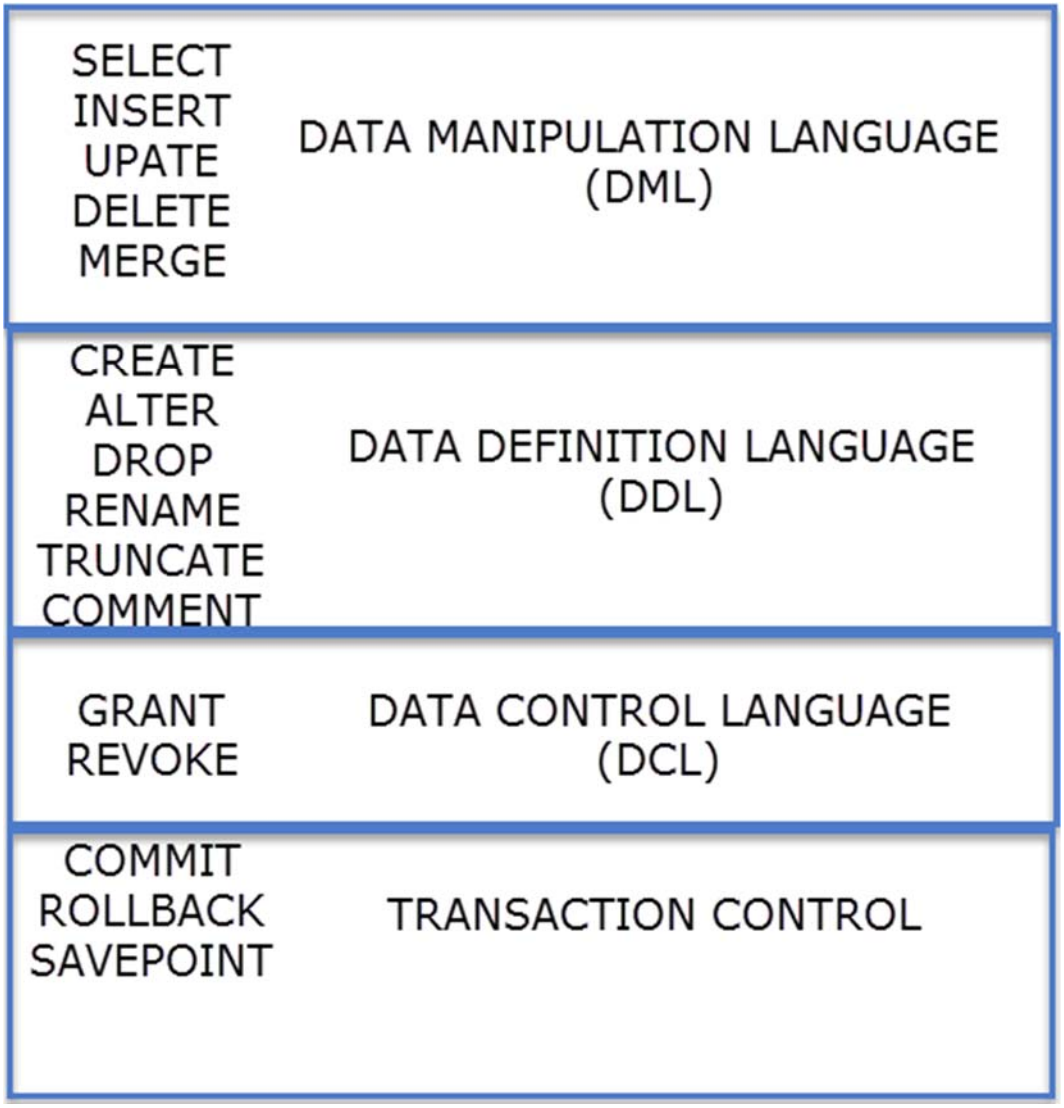
ONLY Oracle should be used for these exercises.

Introduction

Every database has a special interface between end-users and the database engine. In Oracle, this special interface is called SQL*Plus. The basic function of the interface is to allow users to execute commands to the database engine.

You will connect to your database, create a user account and use run the DDL statements to create your database objects from this account.

Once you have created your database objects, you will use DML statement to load some data into the tables and you will be able to retrieve the data by using simple SQL select statements.



Objective

- Use SQL*Plus to connect to an oracle database
- Use dynamic SQL
- Create a user account
- Create a simple table
- Create a complex table
- Insert data into a table
- Retrieve data from a table
- Create a database from a script
- Capture output to a log file.

Tools Used

Databases are carefully planned and designed after gathering and analyzing user requirements from stakeholders.

After the requirements, have been gathered and defined, a model will be created for the database.

This model will be used to use to create the Entity Relationship Diagrams (ERD) and these ERDs will then be translated into Data Definition Languages (DDL).

The DDL script will be used to create your database tables and the DML script will be used to load some data into your tables.

There are some tools to help you create the necessary database tables. These tools include an editor (notepad in your case) and SQL*Plus, and capabilities to print your source code and capture the output.

You will learn how to use these tools in this workbook.

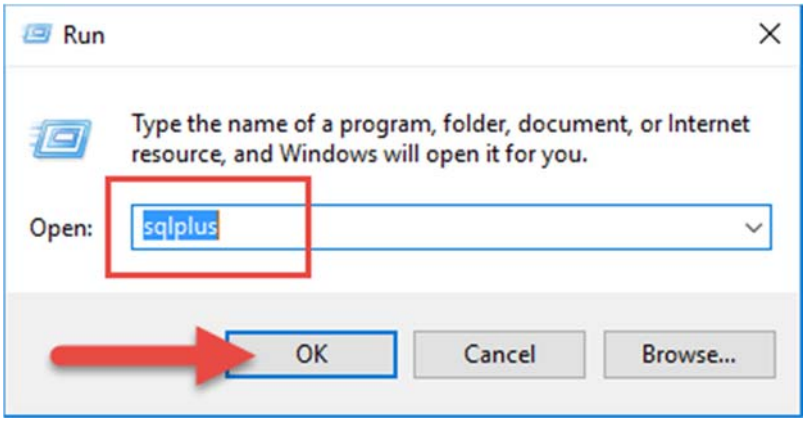
Create a Simple Table Example!

To create a simple table dynamically in SQL*Plus, open sqlplus and run the CREATE TABLE command. In the example below, there will be only one column on the table. You will create the table, then use the describe command (DESC) to be sure the table was created successfully.

Then, while still in sqlplus, enter some data using the INSERT command.

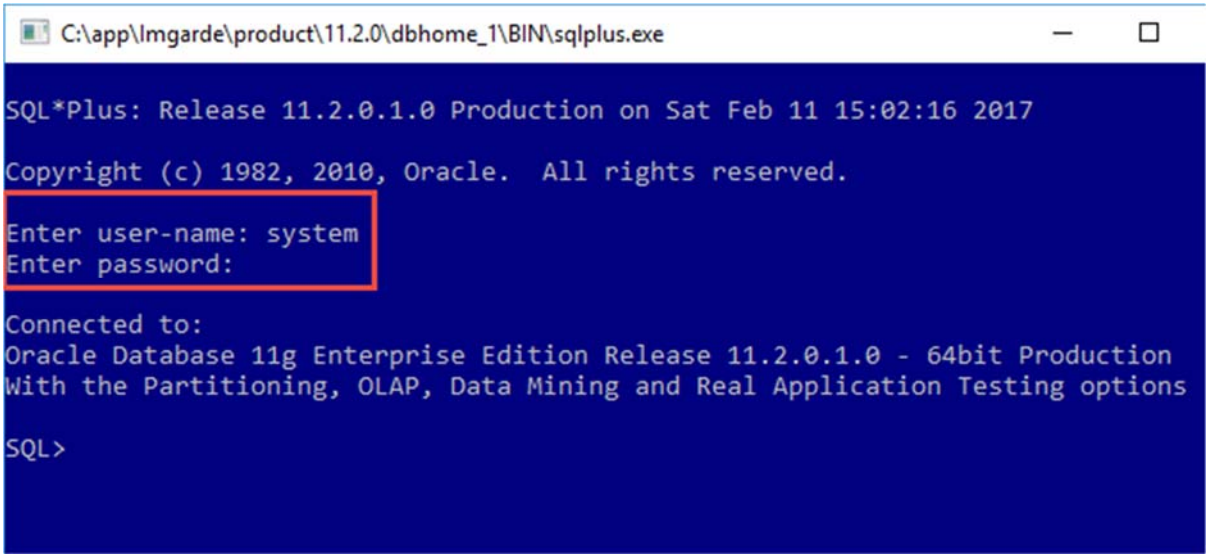
1. Launch SQL*Plus

Right click on the start menu and select Run, then enter sqlplus in the box.



Enter your username/password at the sql prompt.

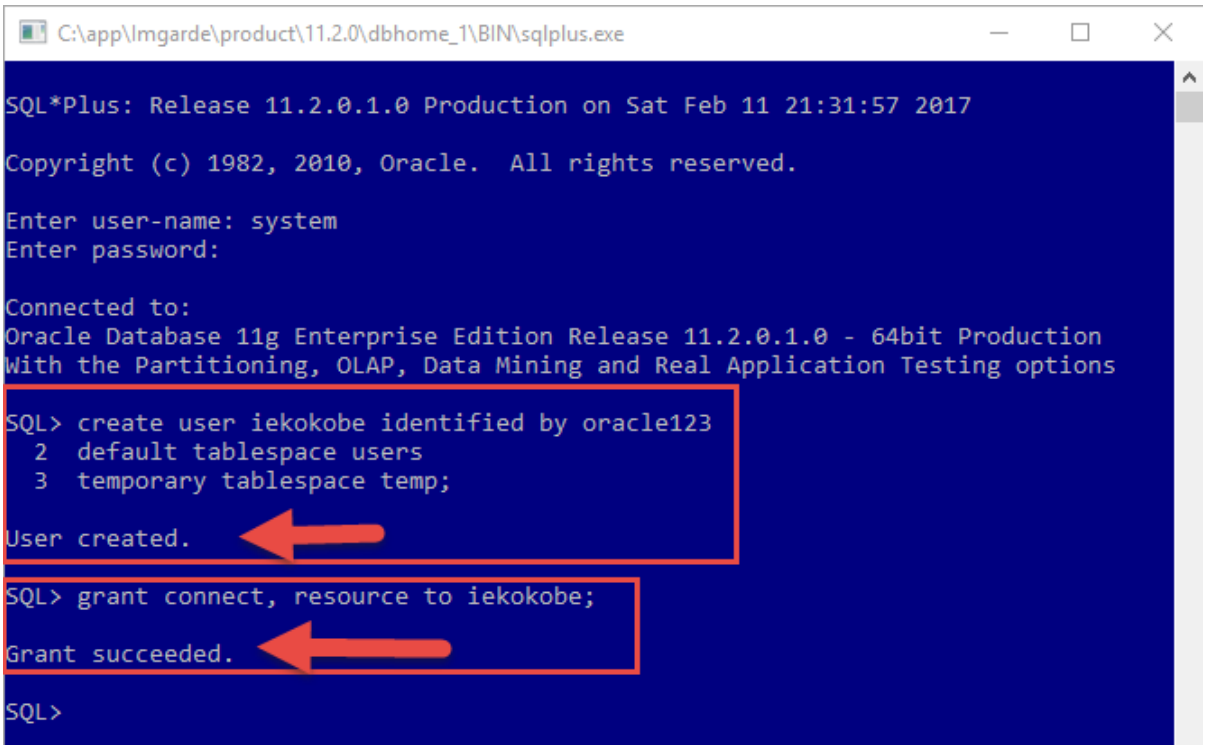
```
SQL*Plus: Release 11.2.0.1.0 Production on Sat Feb 11 11:16:54 2017

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Enter user-name: system
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL>
```



2. Create a user account and give privileges to allow user to connect to the database

```
SQL> create user iekokobe identified by oracle123
  2  default tablespace users
  3  temporary tablespace temp;

User created.

SQL> grant connect, resource to iekokobe;

Grant succeeded.
```

Note: Use your firstname_inital and lastname to create user account (We will learn more about creating user accounts later)



3. Connect to your database from this new account.

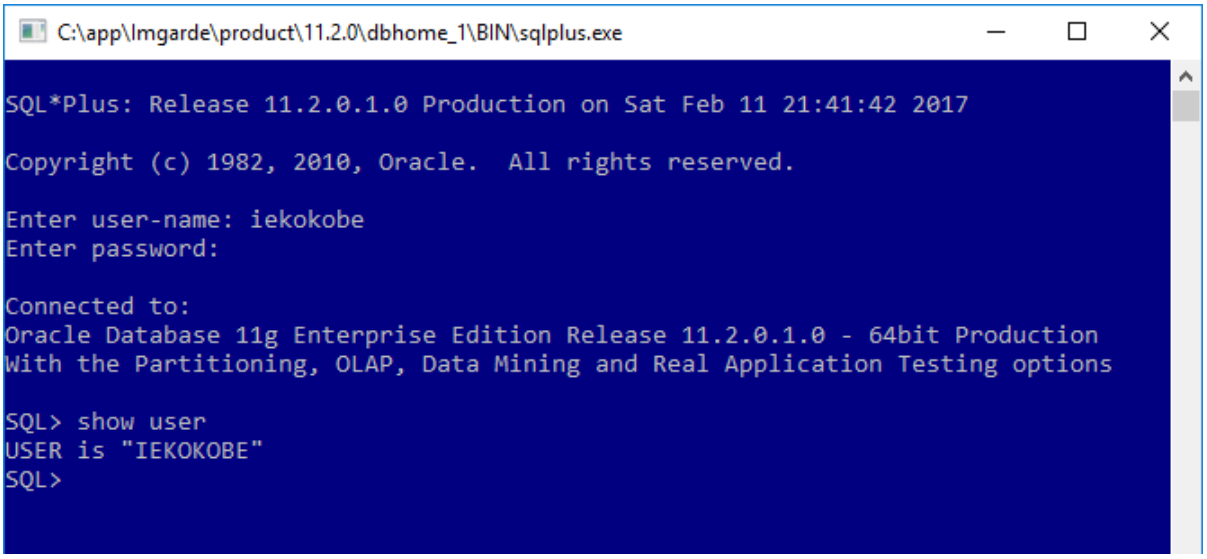
SQL*Plus: Release 11.2.0.1.0 Production on Sat Feb 11 21:39:19 2017

Copyright (c) 1982, 2010, Oracle. All rights reserved.

Enter user-name: iekokobe
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> show user
USER is "IEKOKOBE"
Note: You will create your tables from this account.



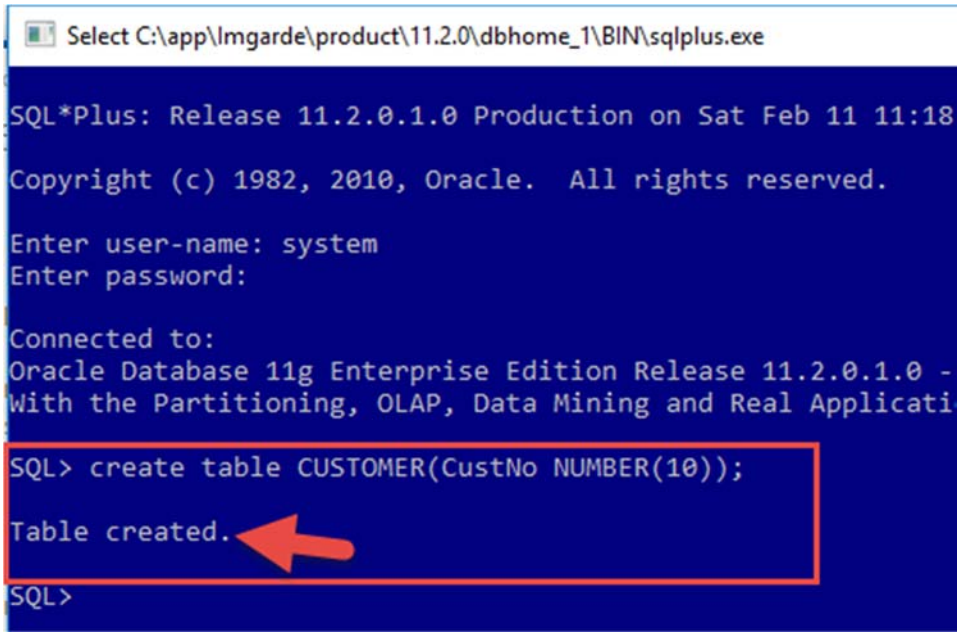
4. Create a simple table

Now that you are connected to your database, you will create a simple table by issuing the CREATE TABLE command as follows:

```
SQL> create table CUSTOMER(CustNo NUMBER(10));

Table created.
```

Oracle will tell you that the table has been created by displaying the Table created output.

Note that the CREATE TABLE statement will not get executed until you add the semi-colon (;) or forward slash (/) at the end.



5. Check to be sure the table was created as specified

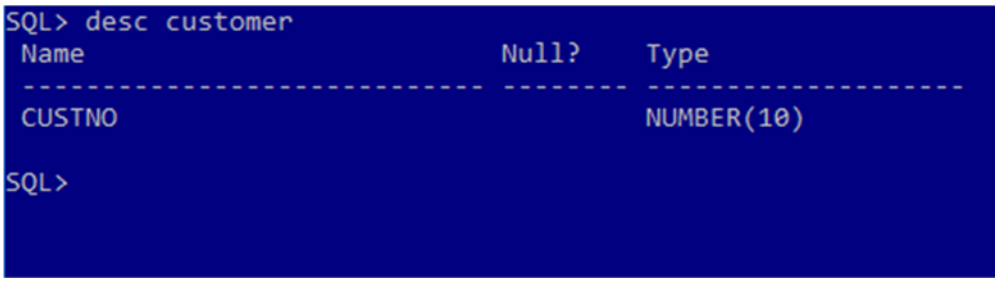
You created your customer table in step 4, now you must verify that it was successfully created within the database.

To do that, you use an sqlplus command called **DESCRIBE** or **DESC** (short form).

Syntax for Describe command:

DESCRIBE table_name

```
SQL> desc customer
 Name                      Null?    Type
 ------------------------- -------- -------------------
 CUSTNO                             NUMBER(10)
```



The output of the results shows that the table was created and the name of the column is CUSTNO and it is of NUMBER datatype with a length of 10.

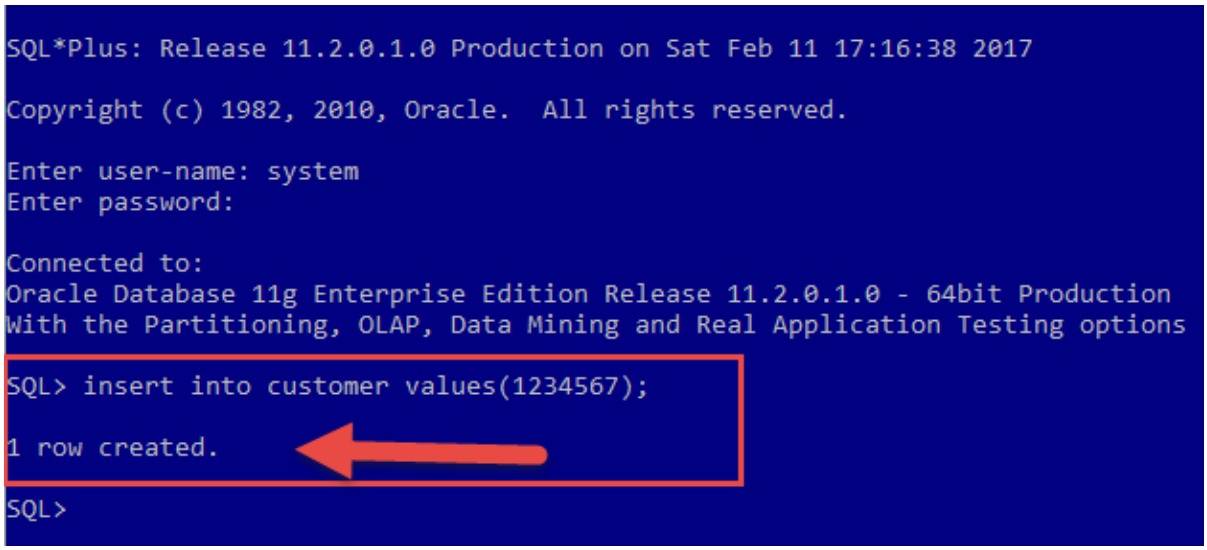
Notice that even though the column was created as mixed case (CustNo), it is displayed in upper case (CUSTNO).

6. Insert data into the customer table.

Your table has been created and you can add a row to it by using the INSERT command as shown below:

```
SQL> insert into customer values(1234567);

1 row created.
```

Oracle acknowledges that the row was inserted correctly by displaying the 1 row created message.

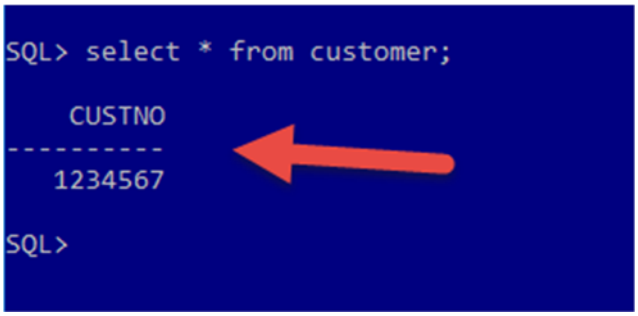


7. Retrieve the data.

You have inserted one row into the customer table in step 6. To confirm the row is there, retrieve it by using a SELECT statement as shown below.

```
SQL> select * from customer;

   CUSTNO
----------
1234567
```

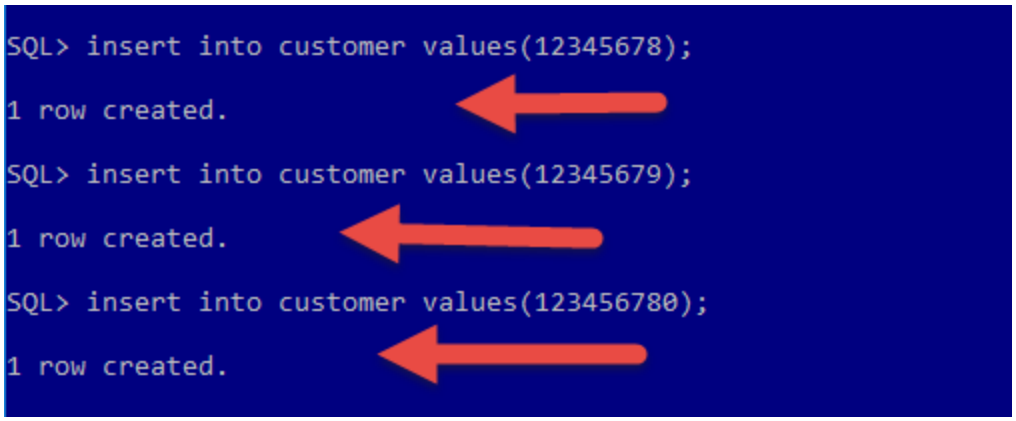
Oracle displays data in the row that you inserted.



6. **Insert more rows**

Now let's add three more rows into the customer table. You do that by using multiple insert commands to add the rows as shown below.

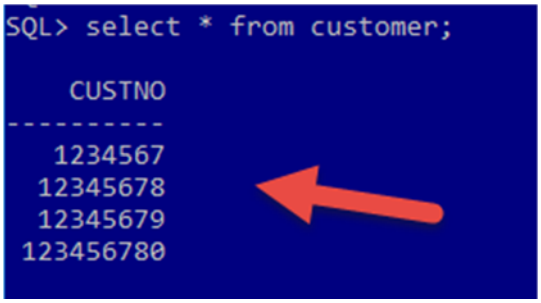
```
SQL> insert into customer values(12345678);

1 row created.

SQL> insert into customer values(12345679);

1 row created.

SQL> insert into customer values(123456780);

1 row created.
```



Now let's retrieve the rows to confirm they are there-

SQL> select * from customer;

CUSTNO ← this CUSTNO represents the column name.
---------- ← this underbar separates the column name from the data
1234567 ← this is the first row of data
12345678 ← this is the second row of data
12345679 ← this is the third row of data
123456780 ← this is the fourth row of data



7. Commit the data.

Now there are four rows in the customer table and we have confirmed they are there by retrieving them.

You are the only person who can see this data, other database users will not be able to see these new data in the database until you commit the change.

You commit the data you have inserted into the customer data by issuing the COMMIT statement.

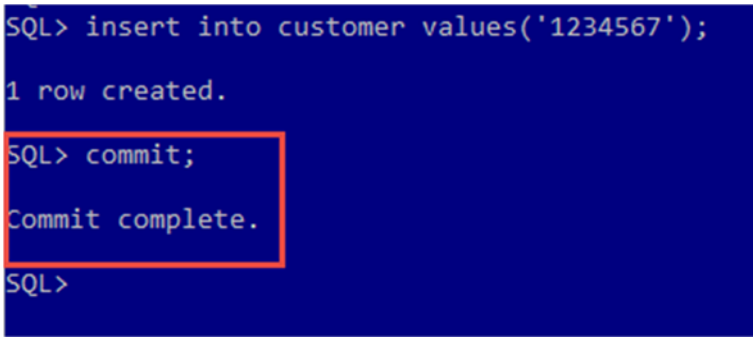
The COMMIT statement ends the current transaction and makes the changes permanent performed in the transaction.

You recall that, a transaction is a series of SQL statements that Oracle treats as a single unit.

If you do not commit the changes and exit your session, all these changes will be lost.

SQL> commit;

Commit complete.



Create a Complex Table Example!

You will create a more complex table with three columns, you will insert 4 rows in this table and retrieve the data in the table just like you did in the example above.

The SQL is a little more complex but the process for creating the table is same.

1. Launch SQL*Plus and connect to your database.

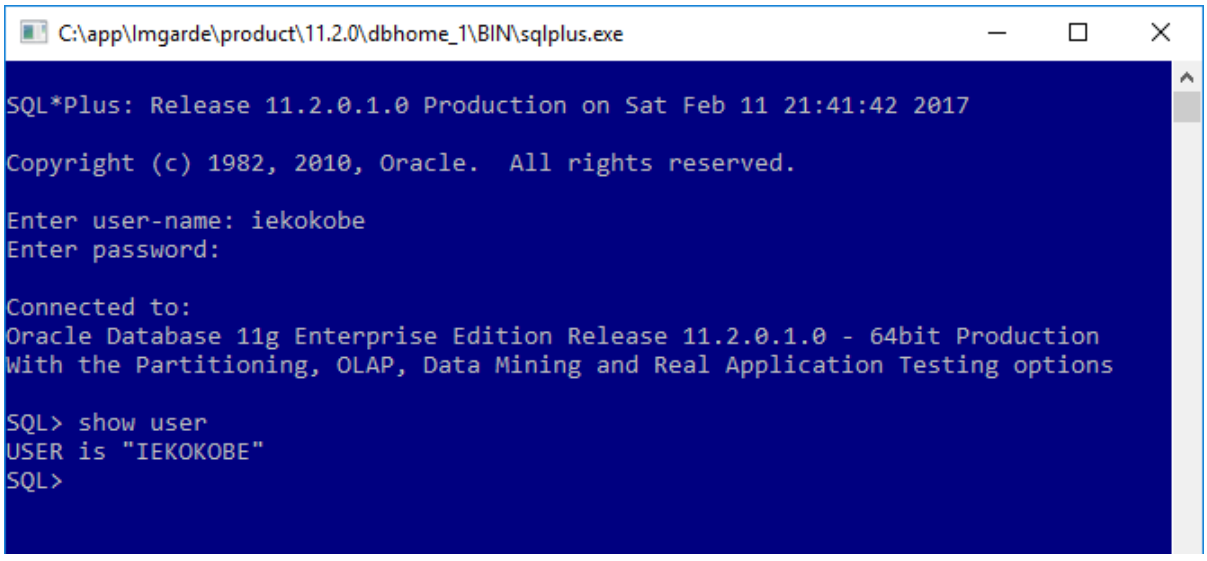
SQL*Plus: Release 11.2.0.1.0 Production on Sat Feb 11 21:39:19 2017

Copyright (c) 1982, 2010, Oracle. All rights reserved.

Enter user-name: iekokobe
Enter password:

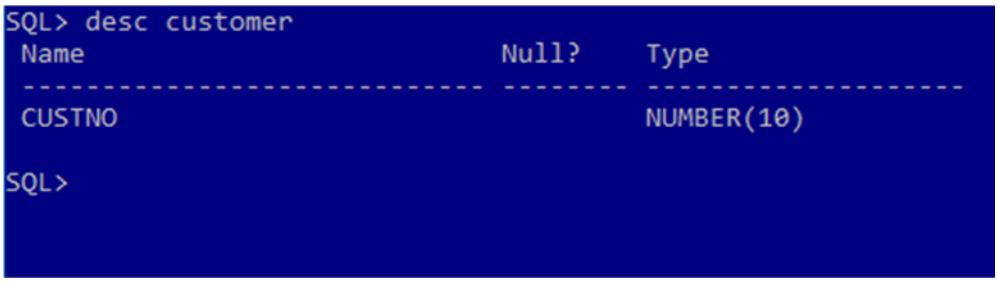
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> show user
USER is "IEKOKOBE"
Note: You will create your tables from this account.



2. Verify that your customer table is there.

```
SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTNO                                             NUMBER(10)
```



3. Drop the CUSTOMER table.

You will drop the customer table you already created because you will recreate it with three columns.

To drop a table in oracle, you use the **DROP TABLE** statement.

DROP TABLE Syntax:

DROP TABLE table_name

Drop the customer table now -

```
SQL> drop table customer;

Table dropped.
```

```
SQL*Plus: Release 11.2.0.1.0 Production on Sat Feb 11 12:24:09

Copyright (c) 1982, 2010, Oracle.  All rights reserved.

Enter user-name: system
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64b
With the Partitioning, OLAP, Data Mining and Real Application T

SQL> desc customer
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------
 CUSTNO                                             NUMBER(10)

SQL> drop table customer;

Table dropped.
```

4. Create a complex customer table.

This time, three columns are defined for the table. Notice that CustNo was defined as not null, which means that a row cannot be inserted unless there is a value specified for that column. That is, the column cannot be blank or empty.

There are also columns for CustFirstName and CustLastName with datatype defined as varchar2 which means data of a variable length character string will be stored of maximum length 15.

```
SQL> create table CUSTOMER
  2  ( CustNo NUMBER(10) NOT NULL,
  3    CustFisrtName   VARCHAR2(15),
  4    CustLastName  VARCHAR2(15)
  5  );

Table created.
```

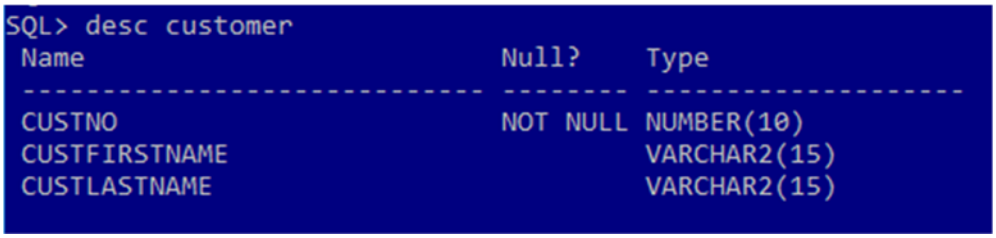
```
SQL> create table CUSTOMER
  2  ( CustNo NUMBER(10) NOT NULL,
  3    CustFirstName    VARCHAR2(15),
  4    CustLastName  VARCHAR2(15)
  5  );

Table created.
```

Note that the CREATE TABLE statement will not get executed until you add the semi-colon (;) or forward slash (/) at the end.

5. Confirm the table was created as expected.

To confirm that the table was created as expected, the describe command or desc is used.

```
SQL> desc customer
 Name                      Null?    Type
 ------------------------- -------- ----------------------
 CUSTNO                    NOT NULL NUMBER(10)
 CUSTFIRSTNAME                      VARCHAR2(15)
 CUSTLASTNAME                       VARCHAR2(15)
```



6. Insert data into the table.

Inserting data is a little more complex because two different data types were used. That is, the varchar2 and number data types were used.

To insert data with the varchar2 data type, the values are put in single quotes. The number data types have no quotation marks.

```
SQL> insert into customer values(1234567, 'kofi', 'woananu');

1 row created.

SQL> insert into customer values(12345678, 'imgard', 'ekokobe');

1 row created.

SQL> insert into customer values(12345679, 'elonge', 'musinga');

1 row created.

SQL> insert into customer values(123456780, 'marvel', 'mobit');

1 row created.
```

```
SQL> insert into customer values(1234567, 'kofi', 'woananu');

1 row created.

SQL> insert into customer values(12345678, 'imgard', 'ekokobe');

1 row created.

SQL> insert into customer values(12345679, 'elonge', 'musinga');

1 row created.

SQL> insert into customer values(123456780, 'marvel', 'mobit');

1 row created.
```

7. Retrieve the data.

Retrieving the data is more interesting with more columns. We can retrieve all the columns and all the rows with SELECT statements.

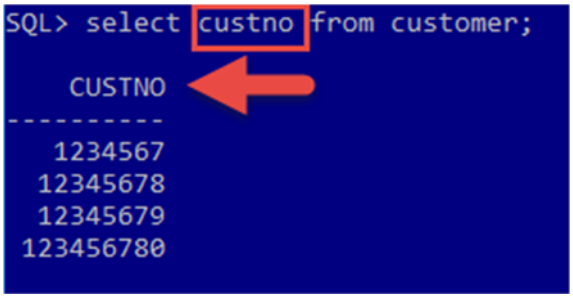
SQL> select * from customer;

CUSTNO	CUSTFISRTNAME	CUSTLASTNAME
1234567	kofi	woananu
12345678	imgard	ekokobe
12345679	elonge	musinga
123456780	marvel	mobit

```
SQL> select * from customer;

    CUSTNO CUSTFISRTNAME   CUSTLASTNAME
---------- --------------- ---------------
   1234567 kofi            woananu
  12345678 imgard          ekokobe
  12345679 elonge          musinga
 123456780 marvel          mobit
```

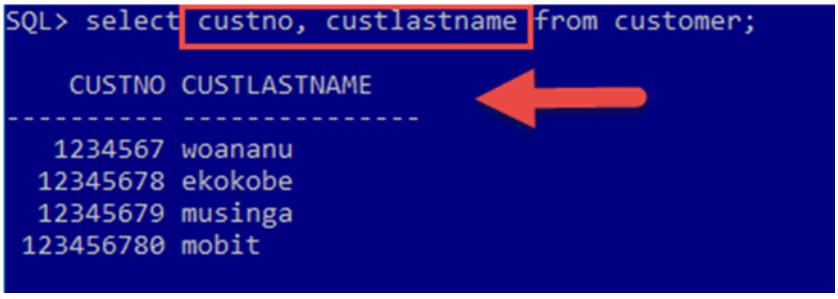
One column can be retrieved as shown below:

```
SQL> select custno from customer;

   CUSTNO
----------
  1234567
 12345678
 12345679
123456780
```



Two columns can be retrieved as shown below:

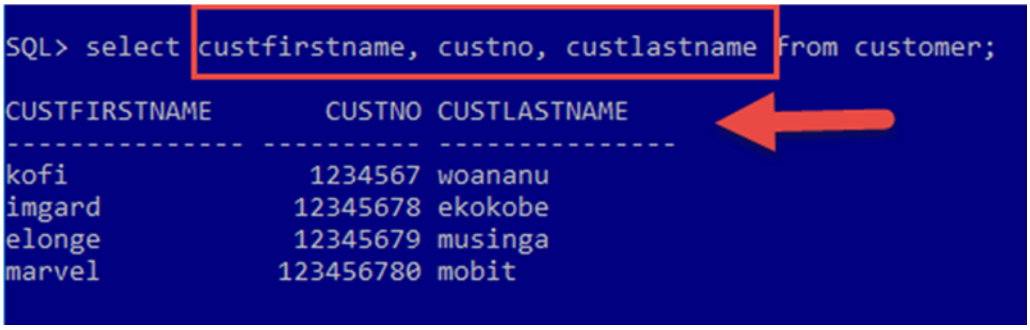
```
SQL> select custno, custlastname from customer;

   CUSTNO CUSTLASTNAME
---------- ---------------
  1234567 woananu
 12345678 ekokobe
 12345679 musinga
123456780 mobit
```



Columns can be displayed in a different order than they are physically stored.

SQL> select custfirstname, custno, custlastname from customer;

CUSTFIRSTNAME	CUSTNO	CUSTLASTNAME
kofi	1234567	woananu
imgard	12345678	ekokobe
elonge	12345679	musinga
marvel	123456780	mobit



Retrieval of the data through SQL can be very powerful. Different combinations of queries can display the data in a variety of ways. The flexibility of SQL provides a very powerful tool for Oracle DBAs.

Running a Script in SQLPLUS

So far, we have executed the DDL statements to create the tables dynamically within SQL*Plus.

On the job, these database objects are not usually created dynamically. You will be provided with scripts which you will review, recommend changes and finalize them with the database developers.

What are the benefits of Using Scripts Versus entering the DDL commands on the command line create your database tables one at a time?

- The script will serve as documentation on the characteristics of the objects.
- Any errors in the syntax can be fixed and the script re-run until it is clean or free of errors.
- The script can be re-used or replicated across the database environments without any issues since they would already have been tested and re-tested.

How Create Tables Using A Script

1. Enter the sql statements in a text file (Your create table statements that were sent to you have already been entered in a text file).
2. Save the file with the *.sql file extension (Your create table statements that were sent to you have already been entered in a text file and saved with the *.sql file extension).
3. Save this file to a folder on your machine. You may name the folder dba_scripts. This is where all your sql scripts will be stored.
4. Open up SQL*Plus
5. Run the script by typing "@filename" at the sql> prompt. For example if my DDL script is located under the C:\Users\Imgarde\Documents\dbascripts folder. I will type the following at the sql> prompt

   sql> @C:\Users\Imgarde\Documents\dbascripts\ unidb.sql

   Note: There should be **NO SPACES** between the @ and the filename when running the script.

Three Ways To Capture The Output Of The DDL Script

1. Screen Capture

The simplest way to capture the output on the screen is to highlight it with the mouse. Then, copy, paste it to a text file (notepad) and save it.

2. Editor

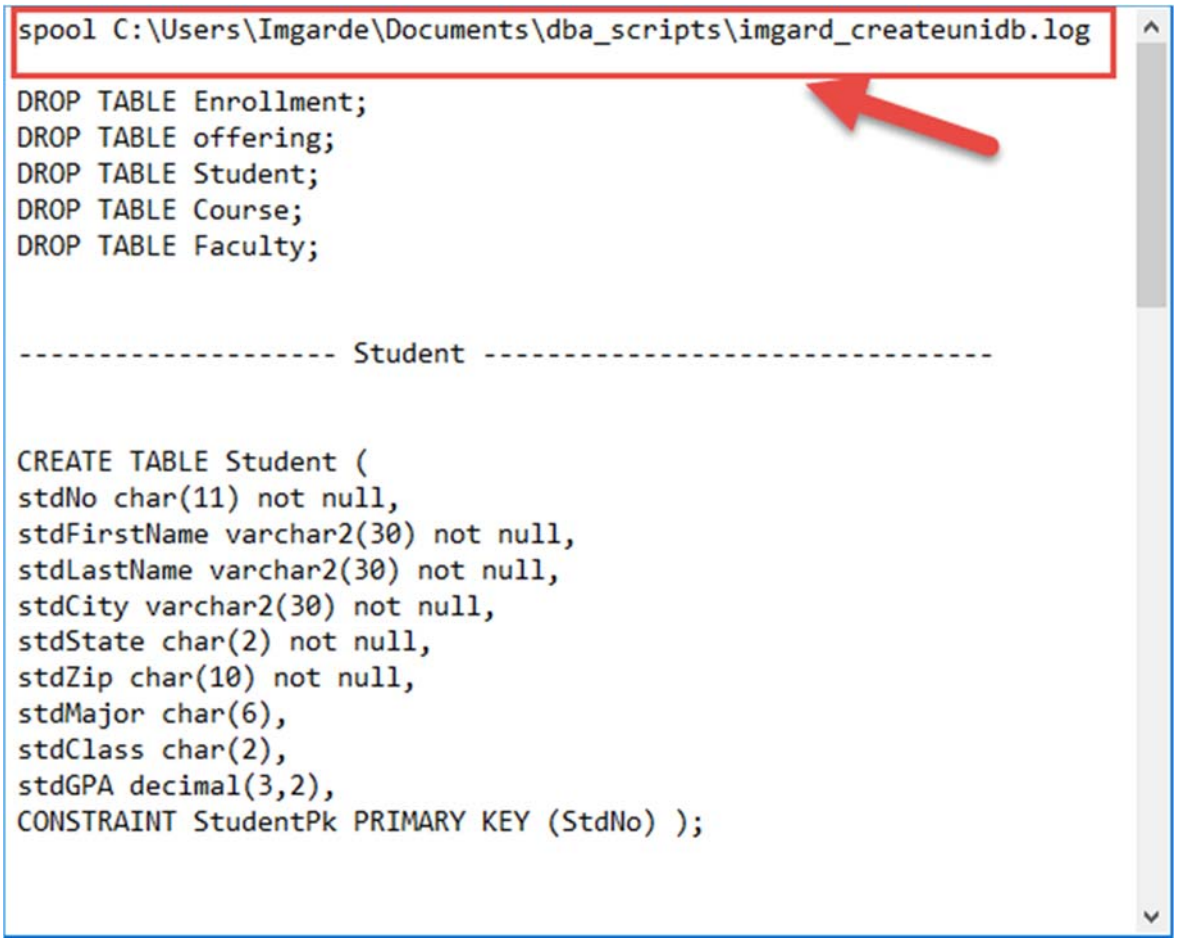
Many editors can be used for database work. However, the editor should have the ability to create a generic text file. A generic text file does not contain special control characters, formatting specific to the editor, etc. In a Unix/Linux environment, the vi editor will be used. On your windows machine, the Notepad editor will be used.

3. Printing the output to a file using the Spool Command

This command is used to record all of the screen displays that would normally run and put them in a file. It is useful to capture the screen output or displays to verify that everything runs correctly.

Two ways to run the SPOOL command to log the output of your DDL (create table script) and DML (script to insert the data in your tables) scripts:

a. **You can type the SPOOL command within the DDL script.**



```
spool C:\Users\Imgarde\Documents\dba_scripts\imgard_createunidb.log

DROP TABLE Enrollment;
DROP TABLE offering;
DROP TABLE Student;
DROP TABLE Course;
DROP TABLE Faculty;


-------------------- Student --------------------------------


CREATE TABLE Student (
stdNo char(11) not null,
stdFirstName varchar2(30) not null,
stdLastName varchar2(30) not null,
stdCity varchar2(30) not null,
stdState char(2) not null,
stdZip char(10) not null,
stdMajor char(6),
stdClass char(2),
stdGPA decimal(3,2),
CONSTRAINT StudentPk PRIMARY KEY (StdNo) );
```

b. You can type the SPOOL command directly from the DDL script **(Preferred method for this assignment)**

☺ At the SQL prompt, you will type the following command which will start printing the output to a file called imgard_createunidb.log in the directory you specified.

In your case, the log file will be called your **firstname_createunidb.log**

SQL> spool C:\Users\Imgarde\Documents\dba_scripts\imgard_createunidb.log

☺ At the next SQL prompt, you will type the following command to run the DDL script to create your 5 tables.

SQL> @C:\Users\Imgarde\Documents\dba_scripts\unidb.sql

☺ At the next SQL prompt, you will type the following command to run the DML script to load the tables with the data.

SQL> @C:\Users\Imgarde\Documents\dba_scripts\unidbload.sql

☺ At the next SQL prompt, you will type the spool off command to stop print to the file.

SQL> spool off

Scripts To Run For Assignment

a. **Set up the script.**

The script provided has already been tested and are free of errors. You are ONLY required to review and run them.

NO CHANGES SHOULD BE MADE TO THESE SCRIPTS

b. **DROP OBJECTS**

Notice the series of DROP table commands at the beginning of the script. Since you may have to run the script more than once, these DROP commands are used to ensure that you start with a clean database.

c. **CREATE TABLES**

Next, you have the CREATE TABLE statements which will create the five tables in your database. When a table is created, space is allocated on disk for that object.

As you can see, the data type of each column is specified.

A decision has to be made about whether or not the columns can be left empty during their lives. If they cannot be empty, the keywords "NOT NULL" must be used. If "NOT NULL" is not used, the default is NULL; that is, the column can be empty at any time. For example, when data is inserted, no value must be specified for that column.

d. **CONSTRAINTS**

We will review the constraints on the table later.

Deliverable

☺ Run the DDL script called unidb.sql to create the tables and turn in output that was captured in a file after it was displayed in SQL*Plus. Filename: yourfirstname_createunidb.log

Filename: yourfirstname_createunidb.log

☺ Run the DML script called unidbload.sql to load data into the tables you created and turn in output that was captured in a file after it was displayed in SQL*Plus.

Filename: yourfirstname_createunidbload.log

☺ Verify that the tables were created. Capture the output in a log file and turn in.

Filename: yourfirstname_verifytables.log

☺ Retrieve all rows in each table. Capture the output in a log file and turn in

Filename: yourfirstname_retrievedata.log

Note: Following directions are CRITICAL for this exercise and points will be deducted if all the 4 log files are not submitted. You will earn a total of 100 points for this assignment.