Ahsanullah University of Science and Technology (AUST)
Department of Computer Science and Engineering

LABORATORY MANUAL

Course No. : CSE4126
Course Title: Distributed Database Systems Lab

For the students of 4th Year, 1st semesterof
B.Sc. in Computer Science and Engineering program

TABLE OF CONTENTS

COURSE OBJECTIVES

The main objective of this lab course is to design distributed databases using Oracle's PL/SQL. In this course, students will learn the important topics of PL/SQL, such as – blocks, functions, procedures, triggers etc. They will also achieve knowledge on setting up distributed database on Oracle over a network via database connections. At the end, students will implement their knowledge by reflecting the concepts from the theory course.

PREFFERED TOOL(S)

- PL/SQL

TEXT/REFERENCE BOOK(S)

- Oracle Pl/Sql Programming, bySteven Feuerstein, Bill Pribyl. PublisherShroff Publishers & Distributors, 2005. (4th Edition)
- M. Tamer Özsu. Principles of Distributed Database Systems, 3rd Edition.

ADMINISTRATIVE POLICY OF THE LABORATORY

- ✓ Students must be performed class assessment tasks individually without help of others.
- ✓ Viva for each program will be taken and considered as a performance.
- ✓ Plagiarism is strictly forbidden and will be dealt with punishment.

SESSION - 1

Objective: The main target of this 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.

Introduction to 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.

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.

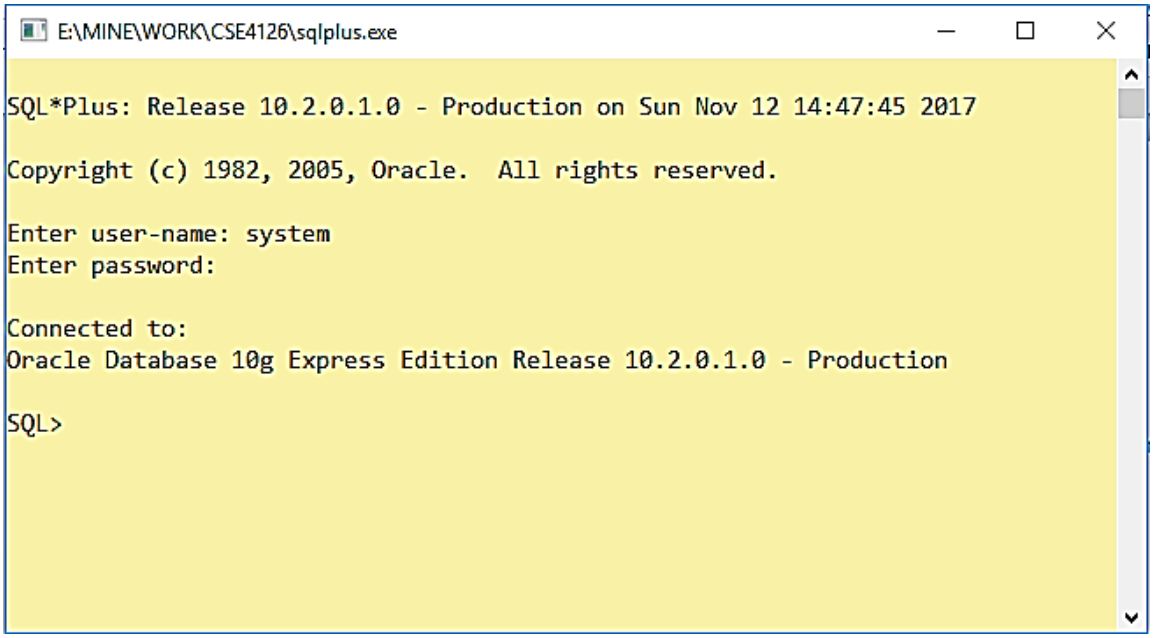
Getting started with SQL*Plus:

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:\oraclexe\app\oracle\product\10.2.0\server\BIN
```

Here you can find `sqlplus.exe`. A convenient way is to copy it to a directory where we want to work.

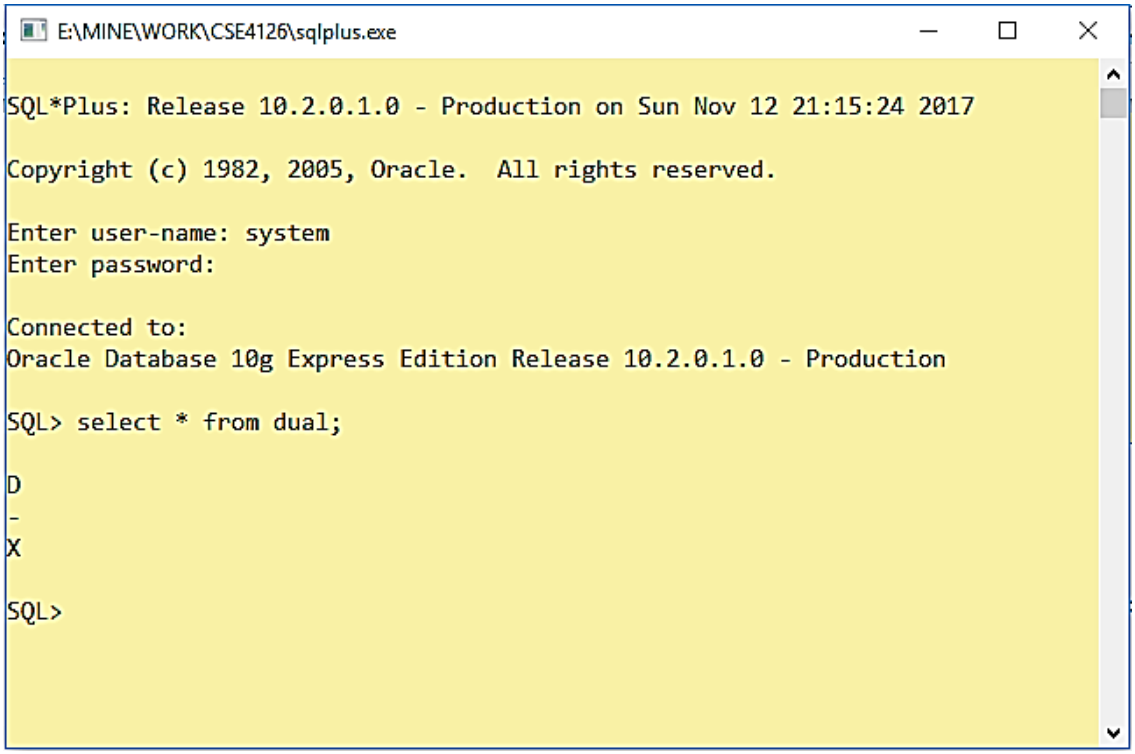
After launching it, enter username and password that you have set in the installation process we enter HR user and its corresponding password.



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.



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');
```

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 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 file `stable.sql` and `insert.sql`. First one contains all the commands to create some tables and second one for inserting data into them. Put them in the same directory of `sqlplus.exe`.

Now enter the command –

```
@./table.sql;

@./insert.sql;
```

Dot (.) means the current directory. You may want to put the files in another folder, say DB, in that case –

```
@./DB/table.sql;

@./DB/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.

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. <u>Join:</u>

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` and `full join`.

2. Sub-query:

```
selectcgpa from student_result
where id = (select id
from student
where name = 'Kavin');
```

3. Set:

```
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.

```
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`.

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.

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 them 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.

```
BEGIN
   NULL;
END;
/
```

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

```
PL/SQL procedure successfully completed.
```

Now, try this –

```
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 –

```
SET SERVEROUTPUT ON;

DECLARE
     n_id student.id %TYPE;
     v_name student.name %TYPE;
BEGIN
     select id, name
     inton_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.

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.

```
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
      inton_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
      inton_id, v_name
      from student
      where id = 3;

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

END;
/
```

Practice Problem 1

Student (snum: integer, sname: string, major: string, slevel: string, age: integer)
Course (cnum: integer, cname: string, meets_at: string, room: string, fid: integer)
Enrolled (snum: integer, cnum: int)
Faculty (fid: integer, fname: string, deptid: integer)

Part – A: For the given schema, perform the following queries (without PL/SQL anonymous block) so that **no duplicates** are printed in any of the answers:

1. Find the names of all students in level - 1 who are enrolled in a course taught by faculty 'Abdullah'.
2. Find the names of students enrolled in the maximum number of courses.

Part – B: For the given schema, perform the following queries (in a PL/SQL anonymous block). Use DBMS_OUPUT.PUT_LINE( ) to print result:

1. Count and print number of students who have not enrolled in any course.
2. Count and print the number of students who are enrolled in two courses that meet at the same time.

SESSION - 2

Objective: The main target of this lab session is to learn –

- PL/SQL Control Statements
- PL/SQL Cursor

PL/SQL IF Statement

The PL/SQL IF statement allows you to execute a sequence of statements conditionally. The IF statement evaluates a condition. The condition can be anything that evaluates to a logical value of true or false such as comparison expression or a combination of multiple comparison expressions. You can compare two variables of the same type or convertible type. You can compare two literals. In addition, a Boolean variable can be used as a condition.

The PL/SQL IF statement has three forms: IF-THEN, IF-THEN-ELSE and IF-THEN-ELSIF.

Example:

```
Declare

     theTakamoney.taka%TYPE := 180;
     theIncmoney.taka%TYPE := 10;

begin
     theTaka := theTaka + theInc;

     iftheTaka< 170
          then
          insert into money values(7,'A',theTaka+10);

     elsiftheTaka< 210 and theTaka>=170
          then
          insert into money values(7,'B',theTaka+30);

     else
          insert into money values(7,'C',theTaka);

     end if;

commit;

end;
/
```

PL/SQL CASE Statement

The PL/SQL CASE statement allows you to execute a sequence of statements based on a selector. A selector can be anything such as variable, function, or expression that the CASE statement evaluates to a Boolean value.

You can use almost any PL/SQL data types as a selector except BLOB, BFILE and composite types.

Unlike the PL/SQL IF statement, PL/SQL CASE statement uses a selector instead of using a combination of multiple Boolean expressions.

Example:

```
declare

     theTaka number := 180;
     theInc number := 10;

begin

     theTaka := theTaka + theInc;

     casetheTaka
     when 170
          then
          insert into money values(7,'new1',theTaka+10);

     when 180
          then
          insert into money values(8,'new2',theTaka+30);

     else
          insert into money values(9,'new3',theTaka);

     end case;

     commit;

end;
/
```

PLSQL Loop Statement

Example:

```
Declare

      theIDmoney.id%TYPE :=0;
      theNamemoney.name%TYPE :='later';
      theTakamoney.taka%TYPE :=100;

begin

      loop

            theID := theID + 1;
            insert into money values(theID,theName,theTaka);
            exit when theID> 10;

      end loop;


end;
/
```

PLSQL Cursor

When you work with Oracle database, you work with a complete set of rows returned from an SELECT statement. However the application in some cases cannot work effectively with the entire result set, therefore, the database server needs to provide a mechanism for the application to work with one row or a subset of the result set at a time. As the result, Oracle created PL/SQL cursor to provide these extensions.

A PL/SQL cursor is a pointer that points to the result set of an SQL query against database tables

Example:

```
setserveroutput on

declare

      theIdmoney.id%TYPE;
      theTakamoney.taka%TYPE;

      cursormoney_cur is
```

```
      selectid,taka from money;

begin

      openmoney_cur;

      loop

      fetchmoney_cur into theId,theTaka;
      exit when money_cur%notfound;

      DBMS_OUTPUT.PUT_LINE(TO_CHAR(theId)||''||TO_CHAR(theTa
ka));

      end loop;

      closemoney_cur;
end;
/
```

Practice Problem 2

Student (snum: integer, sname: string, major: string, slevel: string, age: integer)
Course (cnum: integer, cname: string, meets_at: string, room: string, fid: integer)
Enrolled (snum: integer, cnum: int)
Faculty (fid: integer, fname: string, deptid: integer)

Task: Create a PL/SQL anonymous block that –

A. Prints the last student ID stored in student table.
B. And inserts 10 more tuples in the student table, where IDs will be incremented by 1 starting from the last ID stored (from part - A). Other fields will be just repeated.

Example: Say, we have 5 tuples in student table.

```
SQL> select * from student;

      SNUM SNAME                          MAJ S        AGE
---------- ------------------------------ --- - ----------
         1 Sajid Abdullah                 CSE 1         19
         2 Abdullah Karim                 CSE 1         19
         3 Sajid Rahmatullah              CSE 2         20
         4 Abid Barkatullah               CSE 2         22
         5 Barkatullah Shahid             CSE 3         19
```

After executing your code, the result should be –

```
last id: 5

PL/SQL procedure successfully completed.

SQL> select * from student;

      SNUM SNAME                          MAJ S        AGE
---------- ------------------------------ --- - ----------
         1 Sajid Abdullah                 CSE 1         19
         2 Abdullah Karim                 CSE 1         19
         3 Sajid Rahmatullah              CSE 2         20
         4 Abid Barkatullah               CSE 2         22
         5 Barkatullah Shahid             CSE 3         19
         6 Barkatullah Shahid             CSE 3         19
         7 Barkatullah Shahid             CSE 3         19
         8 Barkatullah Shahid             CSE 3         19
         9 Barkatullah Shahid             CSE 3         19
        10 Barkatullah Shahid             CSE 3         19
        11 Barkatullah Shahid             CSE 3         19

      SNUM SNAME                          MAJ S        AGE
---------- ------------------------------ --- - ----------
        12 Barkatullah Shahid             CSE 3         19
        13 Barkatullah Shahid             CSE 3         19
        14 Barkatullah Shahid             CSE 3         19
        15 Barkatullah Shahid             CSE 3         19

15 rows selected.
```

Note:

1. Use cursor and control statements.
2. Do not use any aggregate functions (i.e. count() function) in your code.
3. Do not use any selection operation inside of the begin-end section.

SESSION - 3

Objective: The main target of this lab session is to learn –

- PL/SQL Function, Procedure

PL/SQL function

PL/SQL function is a named block that returns a value. A PL/SQL function is also known as a subroutine or a subprogram.

Each parameter has one of three modes: IN, OUT and IN OUT.

An IN parameter is a read-only parameter. If the function tries to change the value of the INparameters, the compiler will issue an error message. You can pass a constant, literal, initialized variable, or expression to the function as the INparameter.
An OUTparameter is a write-only parameter. The OUTparameters are used to return values back to the calling program. An OUTparameter is initialized to a default value of its type when the function begins regardless of its original value before being passed to the function.
An IN OUT parameter is read and write parameter. It means the function reads the value from an IN OUT parameter, change its value and return it back to the calling program.
The function must have at least one RETURNstatement in the execution section. The RETURN clause in the function header specifies the data type of returned value.

Example of creating a function:

```
create or replace function isEven(
     n_number in number)
     return number
     is
begin
     if remainder(n_number, 2) = 0
     then
          return 1;
     else
          return 0;
     end if;
```

```
endisEven;
/
```

Example of calling a function:

```
set serveroutput on;

declare
     par number := 10;
     res number;
begin
     res := isEven(par);
     dbms_output.put_line(res);
end;
/
```

We can use a function in a SELECT operation.

```
selectisEven(count(snum)) from student;
```

PLSQL Procedure:

Like a PL/SQL function, a PL/SQL procedure is a named block that does a specific task. PL/SQL procedure allows you to encapsulate complex business logic and reuse it in both database layer and application layer.

Example of creating a procedure:

```
create or replace procedure procAverageAge(

          snumX in student.snum%TYPE,
          snumY in out student.snum%TYPE,
          avgAge out student.age%TYPE)
          IS
begin
          snumY := snumY + 10;

          selectavg(age)
          intoavgAge
```

```
            from student
            wheresnum<snumY and snum>snumX;

end procAverageAge;
/
```

Example of calling procedure:

```
declare

      avgAgestudent.age%TYPE;
      Y student.snum%TYPE := 10;

begin

      procAverageAge(5, Y, avgAge);
      dbms_output.put_line(avgAge);

end;

/
```

Practice Problem 3

Student (*snum*: integer, *sname*: string, *major*: string, *slevel*: string, *age*: integer)

First execute the DB.sql script (provided with the assignment) to create the tables and insert data.

Task [total 10 marks]:

1. Create a PL/SQL **function** that takes a student number as a parameter and returns the number of students of the same age.

2. Create a table as following schema –

age_info (*trackid:* integer*, snum:* integer, *same_age:* number)

   Now, create a PL/SQL **procedure** that –
   a. Takes track id variable as a parameter.
   b. Uses the function created in part – 1 in selection operation and stores the results into a cursor.
   c. Increments track id by 1 and inserts into **age_info** table along with the student numbers and corresponding number of students of the same age (fetched from the cursor).

3. Create a PL/SQL **anonymous block** that invokes the procedure created in – 2 with an initial track id.

Sample output:

For track id 1000 –

```
SQL> select * from student;

      SNUM SNAME                          MAJ S        AGE
---------- ------------------------------ --- - ----------
         1 Sajid Abdullah                 CSE 1         19
         2 Abdullah Karim                 CSE 1         19
         3 Sajid Rahmatullah              CSE 2         20
         4 Abid Barkatullah               CSE 2         22
         5 Barkatullah Shahid             CSE 3         19
         6 Abid Abdullah                  CSE 3         20
         7 Abdullah Hafiz                 CSE 3         21
         8 Rahmatullah Hafiz              CSE 3         21
        11 Sajid Abdullah                 ME  1         19
        12 Abdullah Karim                 ME  1         20
```

```
SQL> select * from age_info;

   TRACKID       SNUM   SAME_AGE
---------- ---------- ----------
      1001          1          3
      1002          2          3
      1003          3          2
      1004          4          0
      1005          5          3
      1006          6          2
      1007          7          1
      1008          8          1
      1009         11          3
      1010         12          2
```

SESSION - 4

Objective: The main target of this lab session is to learn –

- PL/SQL packages

PL/SQL Packages:

Example of creating package:

```
create or replace package myPackage as

    functionisEven(n_number in number)
        return number;

    procedureprocAverageAge(
        snumX in student.snum%TYPE,
        snumY in student.snum%TYPE,
        avgAge out student.age%TYPE);

end myPackage;
/
```

Example of creating package body:

```
create or replace package body myPackage as

    functionisEven(n_number in number)
        return number
        is

    begin

        if remainder(n_number, 2) = 0
            then
            return 1;
        else
            return 0;
```

```
            end if;

      end isEven;

      procedureprocAverageAge(
            snumX in student.snum%TYPE,
            snumY in student.snum%TYPE,
            avgAge out student.age%TYPE)
            IS
      begin
            selectavg(age)
            intoavgAge
            from student
            wheresnum<snumY and snum>snumX;
      end procAverageAge;

end myPackage;
/
```

Example of calling a package element:

```
declare
      par number := 10;
      res number;
      avgAgestudent.age%TYPE;
begin
      res := myPackage.isEven(par);
      dbms_output.put_line(res);
      myPackage.procAverageAge(1, 9, avgAge);
      dbms_output.put_line(avgAge);
      dbms_output.put_line(par);
end;
/
```

SESSION - 5

Objective: The main target of this lab session is to learn –

- PL/SQL Trigger.
- PL/SQL Exception.

PLSQL Trigger:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
- A database definition (DDL) statement (CREATES, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Example:

```
create or replace trigger trigger_log
after update or delete
of age
on student
for each row

begin
     insert into log_updel
     values(:old.age, :new.age, sysdate);
end;
/
```

PL/SQL Exception

In PL/SQL, any kind of errors is treated as exceptions. An exception is defined as a special condition that changes the program execution flow. The PL/SQL provides you with a flexible and powerful way to handle such exceptions.

PL/SQL catches and handles exceptions by using exception handler architecture. Whenever an exception occurs, it is raised. The current PL/SQL block execution halts, control is passed to a separate section called exception section.

There are two types of exceptions:

- *System exception:* the system exception is raised by PL/SQL run-time when it detects an error. For example, NO_DATA_FOUND exception is raised if you select a non-existing record from the database.
- *Programmer-defined exception:* the programmer-defined exception is defined by you in a specific application. You can map exception names with specific Oracle errors using the EXCEPTION_INIT pragma. You can also assign a number and description to the exception using RAISE_APPLICATION_ERROR.

Example:

```
clear screen;
setserveroutput on;
set verify off;

declare
   sc number := &s;
   v_agestudent.age%TYPE;
   res number;

   error_negative exception;

begin

   select age
   intov_age
   from student
   wheresnum = 1;

   ifsc< 0 then
            raiseerror_negative;
   end if;

   res := v_age/sc;
   dbms_output.put_line(res);

exception

   whenzero_divide then
        dbms_output.put_line('Cannot be divided by zero');

   whenno_data_found then
        dbms_output.put_line('not found');

   whenerror_negative then
        dbms_output.put_line('scale factor cannot ve -ve');
```

```
end;
/
```

Practice Problem 4

Firstly, create following two tables – Points and Distance. Points tables stores Cartesian x and y-coordinates of different points identified by different pid. On the other hand, the Distance table stores different pid and distance between two points.

a. **Points** (**pid** integer, **x** number, **y** number).
b. **Distance** (**pid** integer, **dist** number).

Insert some values in Points table. Keep the Distance table empty at first.

Now perform the following tasks:

1. [5 marks] Create a PLSQL trigger that executes after updating Points. It will insert pid into distance table upon which the tuples have been updated. It will also insert into distance table the Cartesian distance between the old and the updated point.
   Note: Use sqrt() and power() function to calculate Cartesian distance.

2. [3 marks] Create PLSQL anonymous block that asks user to input VX, VY and VPID on SQLPLUS command window and updates x and y values in Points where pid matches with VPID.

3. [2 marks] Execute the PLSQL anonymous block and assure that your trigger is working.
   Note: To avoid messages shown on SQLPLUS after providing user inputs, use the following command (the same way you use set serveroutput on, see sample codes also).

```
set verify off;
```

SESSION –6

Objective: The main target of this lab session is to learn –

- Setting up a distributed database over a network using database link

What you need:

- Oracle 10g
- Virtual machine. You can use any VM software, i.e. VMWare.
- Tutorial video: https://youtu.be/OAnh6H7NfTY. The video tutorial is detailed and has some extra materials which is summarized in this presentation. So go through the video first, then follow this presentation.

Overview:

- We want to access a table at a site (virtual machine) from server (host computer) using a database link.

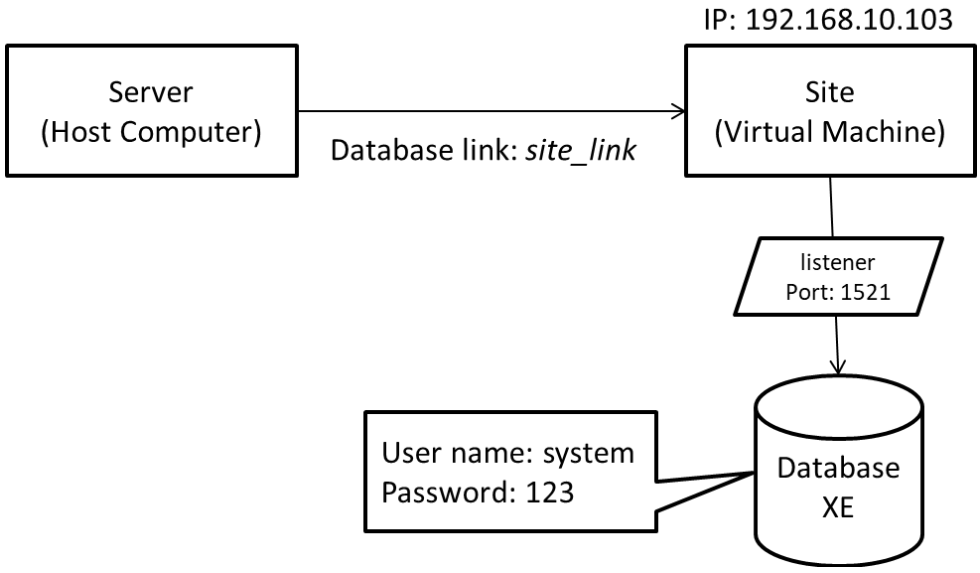


Figure: Overview of distributed database setup on Oracle

Workflow:

Steps:

1. At site:

1.1. Turn off the firewall.
1.2. Get the *IPv4* address (*XXX.XX.XX.XXX*). Note it down.

2. At server: Ping the site from *RUN*. If you get a successful reply, then everything is perfect.

3. At site:

3.1. Go to:
`C:\oraclexe\app\oracle\product\10.2.0\server\NETWORK\ADMIN\`
3.2. Find *listener.ora* file.
3.3. Open listener.ora using NOTEPAD++ and do the following changes.
3.4. Add the following commands to provide additional `SID_LIST` under the `SID_LIST_LISTENER` section:

```
        (SID_DESC =
                (SID_NAME = XE
                (ORACLE_HOME =
                C:\oraclexe\app\oracle\product\10.2.0\server)
          )
```

3.5. Add the following commands to provide additional DESCRIPTION_LIST under the LISTENER section:

```
(ADDRESS = (PROTOCOL = TCP)(HOST = XXX.XX.XX.XXX)(PORT =
1521))
```

3.6. Save the changes.
3.7. Run *CMD* with the administrative mode.
3.8. In *CMD*, run the command `lsnrctl stop`. If you get a successful message then everything is okay.
3.9. Again in *CMD*, run the command `lsnrctl start`. If you get a successful message then everything is okay.

4. At server:

4.1. Run your *sqlplus* and log in. Execute the following codes (also provided as conn.sql) to generate a database link with the site:

```
drop database link site_link;
```

```
create database link site_link
connect to username identified by "password"
using '(DESCRIPTION =
(ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)
(HOST = XXX.XX.XX.XXX) (PORT = 1521)))
(CONNECT_DATA = (SID = XE))
)' ;
```

4.2. Now, *select/ insert/ delete* any data of the site from server using `@site_link`. For example:

```
select * from student@site_link;
```

Practice Problem 5

Design and build a distributed database on two or more computers following the above steps. Apply horizontal fragmentations on the global relations.

Some FAQ's on the Project:

Q: What is the best strategy to do fragments?

A: For doing fragments, you can observe it from the perspective of theory course. You can design good fragmentation schema from the examples and exercises from the textbook. For example, figure 3.4 (page 34), exercise 3.2, 4.3 etc.

Q: How can we allocate or store the fragments in different sites?

A: You have to create table separately at each site. In order to allocate data into the individual site, you can write codes in the server that will insert data into the tables of a site according to the fragmentation schema. Another simple way is to insert data manually. But the first method is very efficient.

Q: Which methods from theory should we implement?

A: The following features can be included in your project.

1. You should be able to simulate level - 3 distribution transparency (as discussed in the theory class). Therefore, you should apply the effect of update steps in your code. [Ch. 3]
2. You can simulate some operator trees for your project. [Ch. 5]
3. You can also implement an operator tree with its canonical expression. For example, if someone wants to SELECT from EMP table, your code will apply canonical expression and extract the result of (EMP1 UN EMP2) instead. It is similar to simulate level - 1 distribution transparency. [Ch. 3 + 5]
4. You can simulate the algebra of qualified relations and proof the rules by applying them to actual table and data. [Ch. 5]
5. You can also apply to estimate profiles of results of algebraic operation by implementing database profiles. [Ch. 6]
6. You can simulate semi-join programs for join queries. [Ch. 6]
7. You can even implement a machine learning technique (i.e. KNN classifier) for your project.

Note that, it is not mandatory to apply all of the above features in your project. The more features you implement, the more positive reviews you gain. You can also implement

something useful, which is not listed above. Implement the features as functions or procedures, and if possible, as packages.

Q: Do we need to implement triggers?

A: Yes. At least two triggers.

Q: Do we need to write the final report?

A: Yes! The report will represent your whole work. Try to make it attractive. Include all the theoretical and practical details (figures, relational algebra, operator trees etc if necessary). While writing about functions and procedures, including their inputs, outputs, and functionalities or description of how it works. No need to provide the screenshots of the results.

MID TERM EXAMINATION

There will be a 40-minutes written mid-term examination. Different types of questions will be included such as MCQ, mathematics, writing code fragments etc.

FINAL TERM EXAMINATION

There will be a one-hour written examination. Different types of questions will be included such as MCQ, mathematics, write a program etc.