Pir Mehr Ali Shah Arid Agriculture University Rawalpindi

University institute of Information Technology

Lab Manual 2020

Teacher: Ms. Sarfraz Bibi

Week 1 2 & 3

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Week 1:

How to open Oracle strcutre query Langauage (sql) ;

Click on start button > Program > Oracle10g/8i-home > Aplication development > click Sqlplus.

Write down user name Scott password tiger and string dba or db.

Creating the following Table in sql:

ID	Name	Class	Section
1	Xyz	BSCS	C
2	Abc	Bsit	B

SQL>create table Class_info (id number(12), name varchar2(10),class char(10), section char (7));

How to Insert data into Table:

SQL>insert into class_info values (1,'xyz','BSCS','C');

1 Row created.

SQL>insert into Class_info values(2,'abc','bsit',B');

1 row created.

Question:1

Create the following table in sqlplus through query?

Student_info

Reg_no	Nicno	Name	Address	Fee	Marks	Grade

Question:2

Insert the data in the above created table?

Writing Basic SQL Statements:

After completing this lesson, you should be able to do the following:

- List the capabilities of SQL SELECT statements
- Execute a basic SELECT statement
- Differentiate between SQL statements and SQL*Plus commands

Writing SQL Statements:

- SQL statements are not case sensitive.
- SQL statements can be on one or more lines.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Tabs and indents are used to enhance readability.

Basic SQL SELECT Statements:

```
SELECT [DISTINCT] {*, column [alias],...}
                FROM   table;
```

- **SELECT identifies *what* columns.**
- **FROM identifies *which* table.**

Selecting All Columns:

```
SQL> SELECT *  FROM    dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Selecting specific Column:

```
SQL> SELECT deptno, loc  FROM   dept;

    DEPTNO LOC
---------- -------------
        10 NEW YORK
        20 DALLAS
        30 CHICAGO
        40 BOSTON
```

Questio#3

Show all data from table Student_info?

Question#4

Show Student name ,reg no ,marks and grade from the table Student_info?

Week #2

Arithmetic Expressions:

Create expressions on NUMBER and DATE data by using arithmetic operators.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
% Modulo	Return the integer remainder of a division .For Example ,12%5=2 because the remainder of 12 divided by 5 is 2.

The plus(+) and minus (-) operator can also be used to perform arithmetic operations on the datetime and smalldatetime value.

SQL> SELECT ename, sal, sal+300 FROM emp;

ENAME SAL SAL+300

---------- -------- ---------

KING 5000 5300

BLAKE 2850 3150

CLARK 2450 2750

JONES 2975 3275

MARTIN 1250 1550

ALLEN 1600 1900

...

14 rows selected.

Operator Precedence:

***,/,+,-**

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to
- clarify statements.

SQL> SELECT ename, sal, 12*sal+100 FROM emp;

ENAME	SAL	12*SAL+100
KING	5000	60100
BLAKE	2850	34300
CLARK	2450	29500
JONES	2975	35800
MARTIN	1250	15100
ALLEN	1600	19300

…

14 rows selected

Using Parenthesis:

SQL> SELECT ename, sal, 12*(sal+100) FROM emp;

```
ENAME              SAL 12*(SAL+100)
---------- --------- -----------
KING              5000       61200
BLAKE             2850       35400
CLARK             2450       30600
JONES             2975       36900
MARTIN            1250       16200
...
14 rows selected.
```

SQL>select

Task :

1. In July 2014 salary of employees increase 800 per head you perform this action in emp table.and show its total annually salary.
2. In every month transport charges are detected from employ salary your task is to show this problem in sql emp table.
3. Find out employ annual salary and minus the old benefits amount from emp salary where old benefit amount is 132.
4. Find out the 15 days salary of employee.
5. Find out total salary of employ if minus the salary of 4 weeks ends.

Defining a Column Alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows column name; optional AS keyword between column name and alias
- Requires double quotation marks if it contains spaces or special characters or is case sensitive

SQL> SELECT ename AS name, sal salary FROM emp;

```
NAME                 SALARY
-------------------- ---------
...
```

SQL> SELECT ename "Name", sal*12 "Annual Salary"
FROM emp;

```
Name                 Annual Salary
-------------------- -------------
```

Concatenation Operator:

- Concatenates columns or character strings to other columns
- Is represented by two vertical bars (||)
- Creates a resultant column that is a character expression.

```
SQL> SELECT        ename||job  AS "Employees"  FROM        emp;


Employees
------------------
KINGPRESIDENT
BLAKEMANAGER
CLARKMANAGER
JONESMANAGER
MARTINSALESMAN
ALLENSALESMAN
...
14 rows selected.
```

Using Literal Character Strings:

```
SQL> SELECT ename    ||' '||'is a'||' '||job    AS "Employee Details"
FROM   emp;
```

Duplicate Rows:

The default display of queries is all rows, including duplicate rows.

```
SQL> SELECT deptno FROM   emp;

  DEPTNO
---------
      10
      30
      10
```

20

...

14 rows selected**.**

Eliminating Duplicate Rows:

Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

SQL> SELECT DISTINCT deptno FROM emp;

DEPTNO

10

20

30

Lab Manual
week # 4 &5

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Order by clause:

Sometimes it is necessary that the rows returned from your query to be in a specific order. For example, I might want scores from high to low or names in alphabetical order. By default, the database will sort data ascending, smallest to largest. Words are sorted alphabetically. NULLs cannot be sorted, they will be listed as found at the bottom of the results.

```
Select ename from employee order by ename;

Select ename from employee order by deptno;

Select ename from emp order by deptno asc;

Select deptno from emp order by deptno desc;
```

Task:

show all data from employee table where name should be in ascending and salary in descending order.

Single Row Function:

Single row functions can be character functions, numeric functions, date functions, and conversion functions. Note that these functions are used to manipulate data items. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Argument can be a column, literal or an expression. Single row functions can be used in SELECT statement, WHERE and ORDER BY clause.

Character functions:

(i) Case conversion function:

Lower(),upper(),initcap()

```
Select lower(ename) from emp ;

Select upper(ename) from emp;

Select initcap(ename) from emp;

SELECT empno, ename, deptno  FROM   emp WHERE
```

```
ename = 'shara';
SELECT empno, ename, deptno from emp  WHERE
    LOWER(ename) = 'shara';
```

(ii) Character manipulation function

Accepts character input and returns number or character value. Functions under the category are CONCAT, LENGTH, SUBSTR, INSTR, LPAD and RPAD.

CONCAT: Joins values together.
(You are limited to using two parameters with CONCAT.)

SUBSTR: Extracts a string of determined length

LENGTH: Shows the length of a string as a numeric value

INSTR: Finds numeric position of a named character

LPAD: Pads the character value right-justified

```
Select concat (deptno,salary) from emp;

Select length(ename) from emp;

Select length(salary) from emp;

Select substr('shara',1,3) from emp where salary <3000
and ename ='shara';

Select * from emp where substr(job,1,7) ='manager';

Select instr('shara','h') from dual;

Select instr(job,'A') from emp;
```

LPAD function pads the left-side of a string with a specific set of characters (when *string1* is not null)

```
LPAD( string1, padded_length, [ pad_string ] )
```

`LPAD('tech', 7);`	would return ' tech'
`LPAD('tech', 2);`	would return 'te'
`LPAD('tech', 8, '0');`	would return '0000tech'
`LPAD('tech on the net', 15, 'z');`	would return 'tech on the net'
`LPAD('tech on the net', 16, 'z');`	would return 'ztech on the net'

```
SELECT LPAD(Last_Name, 10,'*') AS LPAD FROM Employee;

LPAD
----------
****Martin
***Mathews
*****Smith
******Rice
*****Black
```

RPAD function pads the right-side of a string with a specific set of characters (when *string1* is not null).

```
RPAD( string1, padded_length, [ pad_string ] )
```

RPAD('tech', 7);	would return 'tech '
RPAD('tech', 2);	would return 'te'
RPAD('tech', 8, '0');	would return 'tech0000'
RPAD('tech on the net', 15, 'z');	would return 'tech on the net'
RPAD('tech on the net', 16, 'z');	would return 'tech on the netz'

The first query below pads the string 'PSOUG' with the character 'X' up to 10 characters. The second query shows that the padded length takes precedence over the actual length of the original string

```
SELECT RPAD('PSOUG',10,'X') FROM DUAL;

PSOUGXXXXX

SELECT RPAD('PSOUG', 3) FROM DUAL;

PSO
```

Task:

Show the total length of (eno, deptno),joined it and rename by function from the employee table?

Show the ename first 3 alphbet of manager ?

Write a query that show the manager data like this.

```
ename          job
JONES       ***MANAGER

BLAKE       ***MANAGER
```

Number Functions:

```
select round(78.78999,2) from dual;

Select trunc(789.765,2) from dual;

Select mod(com/sal) from emp;

Select mod(6788/678) from dual;

Select sqrt(9) from dual;

Select power(2,3) from dual;
```

Task: Calculate the percentage of any student and round to 0 digit?

Date Functions:

Function	**Description**
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month

The ROUND and TRUNC functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

```
Select MONTHS_BETWEEN(sysdate,hiredate) as "duration"
from employee;

SELECT SYSDATE,
ADD_MONTHS(SYSDATE,1),
ADD_MONTHS(SYSDATE,2),
ADD_MONTHS(SYSDATE,3) ROM  dual;
```

```
SELECT
  SYSDATE,
  NEXT_DAY(SYSDATE,'MONDAY') "Next Mon",
  NEXT_DAY(SYSDATE,'FRIDAY') "Next Fri",
  NEXT_DAY(LAST_DAY(SYSDATE)+1,'TUESDAY') "First Tue"
FROM dual;

SYSDATE   Next Mon  Next Fri  First Tue
--------- --------- --------- ---------
24-JAN-05 31-JAN-05 28-JAN-05 08-FEB-0
```

```
Select Last_Day(sysdate) from employee;
```

```
SELECT
  SYSDATE,
  LAST_DAY(SYSDATE) EOM,
  LAST_DAY(SYSDATE)+1 FOM
FROM dual;

SYSDATE   EOM       FOM
--------- --------- ---------
24-JAN-05 31-JAN-05 01-FEB-05
```

TASk :

```
Show the last date of December 2014?
show the date of next data base class and rename by
cs-400?
```

Conversion Functions:

to_char():

The **to_char()** Oracle conversion function is probably the most commonly used conversion function. The conversion function converts both numerical data and date data to datatype **varchar2.**

```
SELECT   ename, TO_CHAR(hiredate, 'fmDD Month YYYY')
HIREDATE FROM     emp;
```

to_date():

The **to_date()** Oracle conversion function is used to convert character data to the **date** datatype. Like **to_char()**, this Oracle conversion function can be called with a single parameter, much like

```
Select to_date ('02 May 1997', 'DD MONTH YYYY') from
dual;
```

TASK :

```
Convert today date to character?
```

Lab Manual
week #6

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Group function:

A group function is an Oracle SQL function that returns a single result based on many rows, as opposed to single-row functions. These functions are: AVG, COUNT, MIN, MAX, STDDEV, SUM, VARIANCE, etc

Name	Description
Avg()	Return the average value of the argument
Count(Distinct)	Return the count of a number of different values
Count()	Return a count of the number of rows returned
Max()	Return the maximum value
Min()	Return the manimum value
Stddev()	Return the population standard deviation
Sum()	Return the sum

You can use AVG and SUM for numeric data

```
Select avg(sal) from employee ;

Select sum(sal) from employee;
```

You can use MIN and MAX for any datatype.

```
Select max(sal) from employee;
select min(sal) from emp;

Select count(*) from emp;
```

COUNT(*) is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.

```
Select count(distinct deptno) from emp;
Select stddev(salary) from emp;
```

Note: if there is no matching it will returns null value.
Task:show the latest hired employee from employee table?

Group Functions and Null Values:

```
Select avg(comm) from emp;
```

- Group functions ignore null values in the column.
  `SELECT AVG(NVL(comm,0)) FROM   emp;`

Creating groups of data using group by clause:

Divide rows in a table into smaller groups by using the GROUP BY clause.

```
SELECT   deptno, AVG(sal) FROM     emp  GROUP BY deptno;

SELECT   deptno, job, sum(sal) FROM     emp GROUP BY
deptno, job;
```

The sum of all the salaries in the group that you specified in the

GROUP BY clause

The FROM clause specifies the tables that the database must access: the EMP table.

The GROUP BY clause specifies how you must group the rows:

First, the rows are grouped by department number.

Second, within the department number groups, the rows are grouped by job title.

So the SUM function is being applied to the salary column for all job titles within each department number group.

Illegal Queries Using Group Functions:

```
Select deptno,count(ename) from emp;
```

Whenever you use a mixture of individual items (DEPTNO) and group functions (COUNT) in the same SELECT statement, you must include a GROUP BY clause that specifies the individual items (in this case, DEPTNO). If the GROUP BY clause is missing, then the error message “not a single-group group function” appears and an asterisk (*) points to the offending column.

```
SELECT    deptno, AVG(sal)FROM   emp  WHERE
 AVG(sal) > 2000 GROUP BY   deptno;
```

You cannot use the WHERE clause to restrict groups.

You use the HAVING clause to restrict groups.

Using having clause:

Use the HAVING clause to restrict groups

Rows are grouped.

The group function is applied.

Groups matching the HAVING clause are displayed.

```
SELECT   deptno, max(sal) FROM     emp
GROUP BY deptno
 HAVING   max(sal)>2900;
```

If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

The above example displays the department numbers and average salary for those departments whose maximum salary is greater than $2900.

```
SQL> SELECT    job, SUM(sal) PAYROLL
  2  FROM      emp
  3  WHERE     job NOT LIKE 'SALES%'
  4  GROUP BY  job
  5  HAVING    SUM(sal)>5000
 6  ORDER BY  SUM(sal);
```

Nested group function:

Display the maximum average salary.

```
sql> SELECT   max(avg(sal))
  2  FROM     emp
```

```
  3  GROUP BY deptno;
```

Order of evaluation of the clauses:

1. WHERE clause
2. GROUP BY clause
3. HAVING clause

```
SELECT    column, group_function(column)
FROM      table
[WHERE    condition]
[GROUP BY     group_by_expression]
[HAVING   group_condition]
[ORDER BY     column];
```

Displaying data from multiple tables:

The related tables of a large database are linked through the use of foreign and primary keys or what are often referred to as common columns. The ability to join tables will enable you to add more meaning to the result table that is produced. For 'n' number tables to be joined in a query, minimum (n-1) join conditions are necessary. Based on the join conditions, Oracle combines the matching pair of rows and displays the one which satisfies the join condition.

What is join?

- Oracle JOINS are used to retrieve data from multiple tables. An Oracle JOIN is performed whenever two or more tables are joined in a SQL statement.

  Use a join to query data from more than one table.

  – Write the join condition in the WHERE clause.

  – Prefix the column name with the table name when the same column name appears in more than one table.

Oracle INNER JOIN (or sometimes called simple join)

Oracle LEFT OUTER JOIN (or sometimes called LEFT JOIN)

Oracle RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

Oracle FULL OUTER JOIN (or sometimes called FULL JOIN)

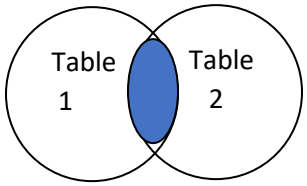
Inner join (Simple join):

It is the most common type of join. Oracle INNER JOINS return all rows from multiple tables where the join condition is met.

General syntax 1:

SELECT ***table1.column, table2.column*** **FROM** ***table1, table2***

WHERE ***table1.column1 = table2.column2***;.



We have a table called *suppliers* with two fields (supplier_id and supplier_ name). It contains the following data:

Supplier_id	Supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have another table called *orders* with three fields (order_id, supplier_id, and order_date). It contains the following data:

Order_id	Supplier_id	Order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10002	2003/05/14

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

OR

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this.

Supplier_id	Supplier_name	Order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13

```
SELECT   emp.empno, emp.ename, emp.deptno,
         dept.deptno, dept.loc
    FROM      emp, dept
    WHERE     emp.deptno=dept.deptno;
```

Using Table Aliases:

```
SELECT e.empno, e.ename, e.deptno,
           d.deptno, d.loc
   FROM   emp e, dept d WHERE  e.deptno=d.deptno;
```

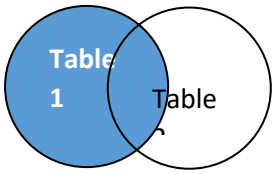
LEFT OUTER JOIN:

Another type of join is called an Oracle **OUTER JOIN**. This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

The syntax for the Oracle **LEFT OUTER JOIN** is:

```
SELECT columns FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

In some databases, the LEFT OUTER JOIN keywords are replaced with LEFT JOIN.



Supplier

Supplier_id	Supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

Order

Order_id	Supplier_id	Order_date
500125	10000	2003/05/12
500126	10001	2003/05/13

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

OR

- **You use an outer join to also see rows that do not usually meet the join condition.**
- **Outer join operator is the plus sign (+).**

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id(+);
```

Our result set would look like this.

Supplier_id	Supplier_name	Order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13
1002	Microsoft	<null>
10003	NVIDIA	<null>

```
SELECT   e.ename, d.deptno, d.dname
  2  FROM     emp e, dept d
  3  WHERE    e.deptno(+) = d.deptno
  4  ORDER BY e.deptno;
```

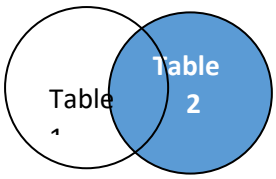
RIGHT OUTER JOIN

Another type of join is called an Oracle **RIGHT OUTER JOIN**. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

The syntax for the Oracle **RIGHT OUTER JOIN** is:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

In some databases, the RIGHT OUTER JOIN keywords are replaced with RIGHT JOIN.



Supplier

Supplier_id	Supplier_name
10000	Apple

10001	Google

Order

Order_id	Supplier_id	Order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10002	2003/05/14

```
SELECT orders.order_id, orders.order_date,
suppliers.supplier_name

FROM suppliers

RIGHT OUTER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;
```

OR

```
SELECT orders.order_id, orders.order_date,
suppliers.supplier_name

FROM suppliers, orders

WHERE suppliers.supplier_id(+) = orders.supplier_id;
```

Our result set would look like this.

order_id	order_date	supplier_name
500125	2013/08/12	Apple
500126	2013/08/13	Google
500127	2013/08/14	<null>

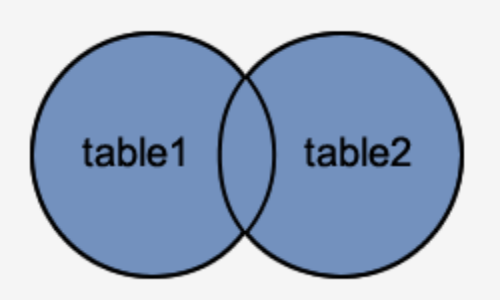
FULL OUTER JOIN

Another type of join is called an Oracle **FULL OUTER JOIN**. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

The syntax for the Oracle **FULL OUTER JOIN** is:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

In some databases, the FULL OUTER JOIN keywords are replaced with FULL JOIN.



This FULL OUTER JOIN example would return all rows from the suppliers table and all rows from the orders table and whenever the join condition is not met, <nulls> would be extended to those fields in the result set.

If a supplier_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set. If a supplier_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

We have a table called **suppliers** with two fields (supplier_id and name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have a second table called **orders** with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10004	2013/08/14

If we run the SELECT statement (that contains a FULL OUTER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date

FROM suppliers

FULL OUTER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	supplier_name	order_date
10000	IBM	2013/08/12
10001	Hewlett Packard	2013/08/13
10002	Microsoft	<null>
10003	NVIDIA	<null>
<null>	<null>	2013/08/14

Joining table to itself:

```
SELECT worker.ename||' works for '||manager.ename
  2  FROM    emp worker, emp manager
  3  WHERE   worker.mgr = manager.empno;


WORKER.ENAME||'WORKSFOR'||MANAG
------------------------------
BLAKE works for KING
CLARK works for KING
JONES works for KING
MARTIN works for BLAKE
...
13 rows selected.
```

Lab Manual

WEEK 7&8

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Week # 7 & 8:

Using subquerries:

Syntax:

```
SELECT      select_list  FROM     table
WHERE       expr operat      (SELECT      select_list   FROM     table);
```

The subquery (inner query) executes once before the main query.
The result of the subquery is used by the main query (outer query).

```
SELECT last_name
FROM   employees
WHERE  salary >
            (SELECT salary
             FROM   employees
             WHERE  last_name = 'Abel');
```

Guide lines for sub query:

Enclose subqueries in parentheses.
Place subqueries on the right side of the comparison condition.
The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis.
Use single-row operators with single-row subqueries and use multiple-row operators with multiple-row subqueries.

```
SELECT last_name, job_id  FROM   employees
   WHERE  job_id =
```

```
                (SELECT job_id
                  FROM   employees
                   WHERE  employee_id = 141);
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

using subquerry in group function:

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

The HAVING Clause with Subqueries

The Oracle server executes subqueries first.

The Oracle server returns results into the HAVING clause of the main query.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
             (SELECT MIN(salary)
              FROM   employees
              WHERE  department_id = 50);s
```

What is wrong with this querry:

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
                (SELECT   MIN(salary)
                 FROM     employees
                 GROUP BY department_id);
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
    (SELECT job_id FROM   employee WHERE
last_name = 'Haas');
```

Task:
Find the employees who earn the same salary as the minimum salary for each

department.

Using the ANY Operator in Multiple-Row Subqueries:

```
SELECT employee_id, last_name, job_id,
salary
FROM   employees
WHERE  salary < ANY
             (SELECT salary
              FROM   employees
              WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';

SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                                (SELECT
mgr.manager_id
                                 FROM
employees mgr);
```

Task:
Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.

Managing Table:

insert,delete,update,merge,control transaction

Copying Rows from Another Table

Write your INSERT statement with a subquery.
Do not use the VALUES clause.
Match the number of columns in the INSERT clause to those in the subquery.

```
INSERT INTO sales_reps(id, name, salary,
commission_pct)
  SELECT employee_id, last_name, salary,
commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';
```

The UPDATE Statement Syntax:

Modify existing rows with the UPDATE statement.
Update more than one row at a time, if required.

UPDATE ***table***
SET ***column = value* [*, column = value, ...*]**
[WHERE ***condition*];**

```
UPDATE    stop11
SET       deptno = 110;
UPDATE    stop11
SET       deptno = 70 where empno=99;
```

Updating Two Columns with a Subquery:

```
UPDATE   stop11
SET  job = (SELECT  job
                FROM    employees
                WHERE   employee_id = 205),
    salary  = (SELECT  salary
                FROM    employees
               WHERE   employee_id = 205)
WHERE    employee_id    =  114;
```

Updating Rows Based on Another Table:

```
UPDATE  copy_emp
SET department_id  =(SELECT
department_id  FROM employee WHERE
employee_id = 100)
WHERE
job_id=(SELECT job_id
FROM employee WHEREemployee_id = 200);
```

The DELETE Statement:

You can remove existing rows from a table by using the DELETE statement.

DELETE [FROM] ***table***

[WHERE ***condition*****];**

```
DELETE FROM departments
 WHERE  department_name = 'Finance';

DELETE FROM  copy_emp;
```

Deleting Rows Based on Another Table:

```
DELETE FROM employees
WHERE  department_id =
                    (SELECT department_id
                     FROM   departments
                      WHERE
department_name LIKE '%Public%');
```

Lab Manual

Week 9

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:	**3(2-3)**	Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Week # 9:

Transaction Control Language (TCL)

Transaction control statements manage changes made by DML statements.
What is a Transaction?
A transaction is a set of SQL statements which Oracle treats as a Single Unit. i.e. all the statements should execute successfully or none of the statements should execute.
To control transactions Oracle does not made permanent any DML statements unless you commit it. If you don't commit the transaction and power goes off or system crashes then the transaction is roll backed.
TCL Statements available in Oracle are
COMMIT :Make changes done in transaction permanent.
ROLLBACK :Rollbacks the state of database to the last commit point.
SAVEPOINT :Use to specify a point in transaction to which later you can rollback.

COMMIT

To make the changes done in a transaction permanent issue the COMMIT statement.
The syntax of COMMIT Statement is
COMMIT [WORK] [COMMENT 'your comment'];
WORK is optional.
COMMENT is also optional, specify this if you want to identify this transaction in data dictionary DBA_2PC_PENDING.
Example

```
insert into emp (empno,ename,sal) values
(101,'Abid',2300);

commit;
```

ROLLBACK

To rollback the changes done in a transaction give rollback statement. Rollback restore the state of the database to the last commit point.
Example :

```
delete from emp;

rollback;            /* undo the changes */
```

SAVEPOINT

Specify a point in a transaction to which later you can roll back.

Example

```
insert into emp (empno,ename,sal) values
(109,'Sami',3000);

savepoint a;
```

```
insert into dept values (10,'Sales','Hyd');
savepoint b;
insert into salgrade values
('III',9000,12000);
```

Now if you give

```
rollback to a;
```

Then row from salgrade table and dept will be roll backed. Now you can commit the row inserted into emp table or rollback the transaction.

If you give

```
rollback to b;
```

Then row inserted into salgrade table will be roll backed. Now you can commit the row inserted into dept table and emp table or rollback to savepoint a or completely roll backed the transaction.

If you give

```
rollback;
```

Then the whole transactions is roll backed.

If you give

```
commit;
```

Then the whole transaction is committed and all savepoints are removed.

Creating and Managing a Table:

- Table Name must begin with a letter
- Can be 1-30 character long
- Must Contain only A-Z, a-z ,o-9,_,$, and #
- Must not duplicate name of another object owned by the same user
- Must not be an oracle reserved word

Creating Table:

```
Create table student_info(id number(2),Name
char(10));
```

Confirming table Creation:

```
Desc student_info;
```

```
Describe student_info;
```

Tables in oracle Database

1. User tables:

- Collection of tables created and maintain by the user
- Contain user information

2. Data dictionary

- Collection of tables created and maintain by the oracle
- Contain database information

```
Select  * from user_tables;
```

Describe tables owned by the user

```
Select Distinct object_type from
user_object
```

View distinct object type owned by the user.

```
Select * from user_catalog;
```

```
Select * from cat;
```

```
Select * from tab;
```

View tables,view,synonyms, and sequences owned by the user.

Creating a Table by using a sub query

- Create a table and insert rows by combining the create table statement and AS subquerry option.
- Match the number of specified columns to the number of subquerry
- Define columns with column names and default value

Syntax:

```
CREATE TABLE table {(column,column……)}
```

```
AS subquerry;
```

Example:

```
Create table dept30 As
select empno,ename ,12*sal as "annual
salary" from emp where deptno=30;
```

Alter table statement:

- Add new column
- Modify an existing column

Syntx:

```
ALTER TABLE table ADD (column
datatype,……);

ALTER TABLE table MODIFY
(columndatatype,……);
```

Example:

Adding column

We use the add clause to add the columns and the new column becomes the last column.

```
Alter table student_info

Add (cell_no number(11));

Alter table student_info add

(address varchar(10),gpa number(2));
```

Modifying a column

We can change a column's data type , size and default value.

```
Alter table student_info

Modify (name varchar(10));
```

Note:

For modifying column should contain null values/empty.

Dropping a Table:

All data and structure in the table is deleted and all indexes are dropped we can't rollback this statement.

```
DROP Table student_info;

Drop table dept;
```

***Changing the name of an Object*:**

To change the name of table,view,sequence,or synonym we execute the rename statement and we must be the owner of the object.

```
RENAME dept  TO department;

RENAME student_info TO std;
```

Truncating a table:

The truncate table statement removes all rows from table release the storage the space used by the tables.

```
TRUNCATE TABLE student_inf;

TRUNCATE TABLE std;
```

NOTE:

We can't roll back row removal when using truncate alternative we can remove rows by using DELETE Statement.

Adding Comments to a Table

We can add comment to a table or column by using the COMMENT statement.

```
COMMENT ON TABLE std
IS  'Arid student information';

COMMENT ON TABLE std | COLUMN std.gpa
IS 'Increase your gpa';
```

Comments can be viewed through data dictionary views

All_col_comments

User_col_comments

All_tab_comments

User_tab_comments

We can drop a comment from the data base by setting is to empty String

```
Comment on table std is ' ';
```

Lab Manual

Week 10

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

Week # 10:

constraints(primary key,foreign key,not null,unique,check)

```
Create table result(sno number(10) ,marks
char(10),gpa number(1,1),

Constraint pk1 primary key (sno));


Create table student(id number(10) primary
key,name char(10) unique,

Cellno varchar(10) not null,sno number(10),

Constraint foreign key fk1 (sno) references
result (sno),

CONSTRAINT std_id_ck

    CHECK (id BETWEEN 10 AND 99));
```

Creating view:

What is view?

You can present logical subset or combination of data by creating views of tables. A view is a logical table based in a table or another view. A view contains no data of its own but is like a window through which data from table can be viewed or change. The tables on which a viewed is based are called base Tables. The view is stored as a SELECT statement in the data dictionary.

Why use views?

- To restrict database access
- To make complex queries easy
- To allow data independence
- To present different views of the same data.

There are two types of view simple views and complex views.

Features	**Simple views**	**Complex views**
Number of table	One	One or more
Contain function	No	Yes
Contains groups of data	No	Yes
DML through views	Yes	Not Always

Creating a view:

You embed a sub query with in the CREATE VIEW statement.
The sub query can contain complex select statement
The sub query cannot contain an ORDER BY clause.

GENERAL SYNTAX:

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW *view*

[(*alias*[, *alias*]...)]

AS *subquery*

[WITH CHECK OPTION [CONSTRAINT *constraint*]]

[WITH READ ONLY]

```
CREATE VIEW empvu10 AS SELECT empno, ename,
job FROM emp WHERE deptno = 10;
```

Using columns aliases:

```
CREATE VIEW   salvu30 AS SELECTempno
EMPLOYEE_NUMBER, ename NAME,sal SALARY FROM
emp WHERE deptno = 30;
```

Modifying a view:

```
CREATE OR REPLACE VIEW
empvu10(employee_number, employee_name,
job_title) AS SELECT empno, ename, job FROM
emp WHERE deptno = 10;
```

Comlex views:

```
CREATE VIEW   dept_sum_vu

 (name, minsal, maxsal, avgsal)

 AS SELECTd.dname, MIN(e.sal), MAX(e.sal),
```

```
 AVG(e.sal)FROM  emp e, dept d

WHERE  e.deptno = d.deptno

  GROUP BY    d.dname;
```

Rules for Performing DML Operations on a View:

You can perform DML operations on simple views.

You cannot remove a row if the view contains the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword

You cannot modify data in a view if it contains:

Any of the conditions mentioned the above Columns defined by expressions.

You cannot add data if:

The view contains any of the conditions mentioned above .

There are NOT NULL columns in the base tables that are not selected by the view.

Using the WITH CHECK OPTION Clause:

```
CREATE OR REPLACE VIEW empvu20

   AS SELECT  *

  FROM emp WHERE  deptno = 20

  WITH CHECK OPTION CONSTRAINT empvu20_ck;
```

```
CREATE OR REPLACE VIEW empvu10
 (employee_number, employee_name, job_title)
   AS SELECT  empno, ename, job
   FROM emp
   WHERE   deptno = 10
   WITH READ ONLY;
```

Removing a View:

Remove a view without losing data because a view is based on underlying tables in the database.

```
Drop VIEW view;

Drop VIEW dept30;
```

Select * from stats;

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.31
13	7	91.7	5.15
44	1	27.3	.18
44	7	74.8	2.11
66	1	6.7	2.1
66	7	65.8	4.52

create a view from above table to convert Fahrenheit to Celsius and inches to centimeters:

whereas celsius=(F- 32) * 5 /9 and centimeter=inches*0.3937

update all rows of table STATS to compensate for faulty rain gauges known to read 0.01 inches.

write a query restricted to January below-freezing (0 Celsius) data, sorted on rainfall:

update one row, July temperature reading, to correct a data entry error where rain in inches is 2.11:

correct reading is 74.9.

Lab Manual

Week 11 &12

Course title :	**Introduction to Database**			Course code:	**Cs-400**
Credit Hours:		Class:	**BS(CS)A/B/C**	Semester:	**4-A,B,C.**

privileges:

Database security:
System security
Data security
System privileges: Gain access to the database
Object privileges: Manipulate the content of the database objects
Schema: Collection of objects, such as tables, views, and sequences

System privileges:

More than 80 privileges are available.

The DBA has high-level system privileges:
Create new users
Remove users
Remove tables
Back up tables

Creating Users:

```
CREATE USER      user
IDENTIFIED BY  password;


CREATE  USER  scott
 IDENTIFIED BY tiger;
```

User System Privileges:

Once a user is created, the DBA can grant specific system privileges to a user.

```
GRANT privilege [, privilege...]
TO user [, user...];
```

An application developer may have the following system privileges:

CREATE SESSION
CREATE TABLE
CREATE SEQUENCE
CREATE VIEW
CREATE PROCEDURE

```
GRANT  create table, create sequence, create
view TO     scott;
```

What Is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes granting and revoking privileges easier to perform and maintain.

A user can have access to several roles, and several users can be assigned the same role. Roles typically are created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax

CREATE ROLE role;

where:roleis the name of the role to be created

Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.

Creating and Granting Privileges to a Role:

SQL>`CREATE ROLE manager;`

```
SQL> GRANT create table, create view
    to manager;
```

```
GRANT   select, insert
  2   ON dept
  3   TO scott
  4   WITH GRANT OPTION;
```

Allow all users on the system to query data from Alice's DEPT table**.**

SQL>`GRANT  select`

```
  2   ON alice.dept
  3   TO PUBLIC;
```

```
GRANT   update (dname, loc)
  2   ON dept
  3   TO scott, manager;
```

```
REVOKE  select, insert
  2   ON dept
  3   FROM   scott;
```

Stored Procedures in PL/SQL

Many modern databases support a more procedural approach to databases—they allow youto write procedural code to work with data. Usually, it takes the form of SQL interweavedwith the more familiar IF statements, etc.
Note that this has nothing to do with accessing the database. You can access any databasefrom virtually any language. What we're talking about is the code that is executed by the database server.
While there are many various 'database' languages, we will only talk about the primarytwo: T-SQL, which is supported by SQL Server and Sybase, and PL/SQL, which is supportedby Oracle.
Many other languages may be supported. For example, Oracle allows you to write storedprocedures and triggers in Java, etc.

PL/SQL Program Blocks

PL/SQL programs are structured in blocks and have the following format:

```
DECLARE
variable_declarations
BEGIN
procedural_code
EXCEPTION
error_handling
END;
```

Declare

The declare part is where variable declaration goes. All used variables must be declared in
this section. This is also the place where other more exotic variable types are declared, like

cursors and exceptions.

Begin

This is the part we're most interested in. This is where the bulk of your programs shall be
placed. Here, you can have IF statements, loops, etc.

Exceptions

The exception section is where we place error handling code.

End

The end signifies the end of this program block.

IF - THEN Structure:

The general format of an IF statement is:
IF condition THEN
program_statements
END IF;
Assuming we all know how to program, and know what IF statements are, I'm not goingto spend too much time on the obvious.
An example program that uses an IF statement is:

```
DECLARE
A NUMBER(6);
B NUMBER(6);
BEGIN
A := 23;
B := A * 5;
IF A < B THEN
```

```
DBMS_OUTPUT.PUT_LINE('Ans: ' || A || ' is less
than ' || B);
END IF;
END;
```

Which produces the expected output of:

```
23 is less than 115.
```

IF ELSIF Structure

When IF and ELSE are not enough, we can resort to using ELSIF. This is an else if equivalentin C (and in Perl it is actually named elsif).

Let's say we wanted to calculate the letter grade given a number grade, we may write a
program such as:

```
DECLARE
NGRADE NUMBER;
LGRADE CHAR(2);
BEGIN
NGRADE := 82.5;
IF NGRADE > 95 THEN
LGRADE := 'A+';
ELSIF NGRADE > 90 THEN
LGRADE := 'A';
ELSIF NGRADE > 85 THEN
LGRADE := 'B+';
ELSIF NGRADE > 80 THEN
LGRADE := 'B';
ELSIF NGRADE > 75 THEN
LGRADE := 'C+';
ELSIF NGRADE > 70 THEN
LGRADE := 'C';
ELSIF NGRADE > 65 THEN
LGRADE := 'D+';
```

```
ELSIF NGRADE > 60 THEN
LGRADE := 'D';
ELSE
LGRADE := 'F';
END IF;
7
DBMS_OUTPUT.PUT_LINE('Grade ' || NGRADE || ' is
' || LGRADE);
END;
```

Which for our particular example number grade produces output:

```
Grade 82.5 is B
```

Example:

```
DECLARE
PID NUMBER(6);
BEGIN
PID := 20;
INSERT INTO product VALUES (PID,'tv',32,199.99);
PID := PID + 1;
INSERT INTO product VALUES(PID,'vcr',16,799.98);
COMMIT;
END;
```

Introduction to Stored Procedures

Just like any other procedural language, PL/SQL has code fragments that are called PROCEDURES.

You can call these PROCEDURES from other code fragments, or directly from SQL*Plus

(or some other client program).

Before you begin to write procedures though, you need to verify that you have enoughprivileges to do that. If you don't (which probably means you're using a plain user account),then you need to login as administrator (or ask the administrator) to grant you access. Togrant such priviledge yourself (in case you're the

administrator - running Oracle on your ownmachine) you can do:

```
GRANT CREATE PROCEDURE TO someusername;
```

From that point on, the user someusername will be allowed to create, drop, and replaceprocedures and functions.

PROCEDURES

Procedures are code fragments that don't normally return a value, but may have some outsideeffects (like updating tables). The general format of a procedure is:
PROCEDURE procedure_name IS
BEGIN
procedure_body
END;
Of course, you'll usually be either creating or replacing the procedure, so you'd want toadd on CREATE (OR REPLACE) to the declaration. For example, to create (or replace) a HELLO procedure, you might do something like this:

```
CREATE OR REPLACE
PROCEDURE HELLO IS
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
```

The above declares a HELLO procedure that just displays 'Hello World'. You can runit as part of a code fragment, or inside other procedures (or functions). For example:

```
BEGIN
HELLO();
END;
```

Or you can simply execute it in SQL*Plus by typing:

```
CALL HELLO();
```

3.8 General Format

The general format of a create procedure statement is this:

```
CREATE OR REPLACE
PROCEDURE procedure_name ( parameters ) IS
BEGIN
procedure_body
END;
```

Where procedure_name can be any valid SQL name, parameters is a list of parameters .

Parameters

The parameters (or arguments) are optional. You don't have to specify anything (not eventhe parenthesis). For example, a sample procedure, which you no doubt have already seen:

```
CREATE OR REPLACE
PROCEDURE HELLOWORLD IS
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

Never actually defines any parameters. What's the use of a procedure that doesn't takeany parameters and doesn't return anything? Well, you may be interested in the procedure's side effects, like in our case, we're interested in our procedure displaying 'Hello World!' and nothing else. There may be many instances where you may want to just do something tothe database, without any particular parameters, and without returning anything.

Anyway, this section is about parameters so let's talk about parameters.Parameters are defined in a similar way as in a CREATE TABLE statement, which is similar to how variables are declared. You first specify the name of the variable, and then the type.

For example:

(N INT)
Would setup some procedure to accept an INT variable named N. Writing a simple procedureto display a variable name, you can come up with something like this:

```
CREATE OR REPLACE
PROCEDURE DISPN (N INT) IS
BEGIN
DBMS_OUTPUT.PUT_LINE('N is ' || N);
END;
```

Which if you call, will promptly display:

```
SQL> CALL DISPN(1234567891);
N is 1234567891
```

You can also have multiple parameters. For example, you can accept A and B and display
their sum and product.

```
CREATE OR REPLACE
PROCEDURE DISP_AB (A INT, B INT) IS
BEGIN
DBMS_OUTPUT.PUT_LINE('A + B = ' || (A + B));
DBMS_OUTPUT.PUT_LINE('A * B = ' || (A * B));
END;
```

Which when ran, displays something like (depending on the values you provide):

```
SQL> CALL DISP_AB(17,23);
A + B = 40
A * B = 391
```

Btw, it should be noted that you can use any PL/SQL type as an argument. For example,
VARCHAR and others are perfectly acceptable. For example:

```
CREATE OR REPLACE
PROCEDURE DISP_NAME (NAME VARCHAR) IS
BEGIN
```

```
DBMS_OUTPUT.PUT_LINE('Hi ' || NAME || '!');
END;
```

Which when called displays:

```
SQL> CALL DISP_NAME('John Doe');
Hi John Doe!
```

3.9 IN, OUT, IN OUT

There are various different parameter varieties (not types). For example, for the time being,we've only been giving the procedure data via parameters. This is the default (IN).
What we could also do is get data from the procedure, via an OUT parameter. To do that,we simply specify OUT in between the parameter name and its type. For example:

```
CREATE OR REPLACE
PROCEDURE SUM_AB (A INT, B INT, C OUT INT) IS
BEGIN
C := A + B;
END;
```

Notice that the above code does not display the resulting sum, it just changes the valueof the C parameter. Also notice the word OUT right after the declaration of C parametername.
Anyway, we will use a code fragment to call the procedure:

```
DECLARE
R INT;
BEGIN
SUM_AB(23,29,R);
DBMS_OUTPUT.PUT_LINE('SUM IS: ' || R);
END;
```

Which when ran, displays:

```
19
SUM IS: 52
```

Notice how we called the procedure with an argument to eventually retrieve the OUTresult.

There is also the other special way of passing parameters: IN OUT. What that means isthat we first can read the parameter, then we can change it. For example, we can write a procedure that doubles a number:

```
CREATE OR REPLACE
PROCEDURE DOUBLEN (N IN OUT INT) IS
BEGIN
N := N * 2;
END;
```

To run it, we also create a small code fragment:

```
DECLARE
R INT;
BEGIN
R := 7;
DBMS_OUTPUT.PUT_LINE('BEFORE CALL R IS: ' || R);
DOUBLEN(R);
DBMS_OUTPUT.PUT_LINE('AFTER CALL R IS: ' || R);
END;
```

Which when ran displays:

```
BEFORE CALL R IS: 7
AFTER CALL R IS: 14
```

Notice how this particular call first grabbed the value of a parameter, then set it in orderto return the double of the value. You can generally intermix these various ways of passing parameters (along with varioustypes). You can use these to setup return values from procedures, etc.

Dropping Procedures

If you're interested in getting rid of a procedure totally, you can DROP it. The general format of a DROP is:

```
DROP PROCEDURE procedure_name;
```