CERTIFICATE

This is to certify that Dhruv Patel

Enrollment .No. 20BEIT30071 of Semester 3, Information Technology Department has satisfactory completed the course in Database Management System – CE 305 / IT 305 at LDRP Institute of Technology and Research, Gandhinagar.

Date of Submission :__________________________________

Concern Faculty :__________________________________

Head of Department : __________________________________

INDEX

LDRP Institute of Technology and Research

Practical 1: Creating and Manipulating Database objects and Applying Constraints (DDL)

Objectives

After completing this lesson, you should be able to do the following:
Describe the main database objects
Create tables
Describe the data types that can be used when specifying column definition
Alter table definitions
Drop, rename, and truncate tables

Creating Tables

```
create table employees(name varchar(20), id number, salary number, post
varchar(20), city varchar(20), country varchar(20));
```

Querying the Data Dictionary

- desc employees;

Creating a Table by Using a Subquery

```
Select enrollment_no,first_name from students where sem=3;
```

Adding a Column

```
alter table employees add(mobile_no number);
alter table employees add(email_id varchar(20));
```

Modifying a Column

```
alter table employees modify post varchar(35);

alter table employees modify name varchar(30);
```

Dropping a Column

```
alter table employees drop column country;
```

Dropping a Table

```
drop table employees_info;
```

Changing the Name of an Object

```
alter table employees rename column id to employe_id;

alter table employees rename to employees_info;
```

Truncating a Table

- truncate table employees_info;

Exercises

1. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.
    - create table DEPT(dept_name varchar2(10), dept_id number);
    - desc DEPT;
    - insert into DEPT(dept_name,dept_id) values('DEEP',10);

- insert into DEPT(dept_name,dept_id) values('JAINAM',1);
- insert into DEPT(dept_name,dept_id) values('RUTVIK',20);

- select * from DEPT;

2. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9_3.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

-
  ```
  create table EMP(emp_name varchar2(20),emp_id number,salary number,emp_branch
  varchar(20),emp_experience number);
  ```

  ```
  desc EMP;
  ```

3. Modify the EMP table to allow for longer employee last names. Confirm your modification.

-
  ```
  alter table EMP add(emp_last_name varchar(20));
  ```

  ```
  desc EMP;
  ```

-
  ```
  alter table EMP modify emp_last_name

  varchar(30); desc EMP;
  ```

4. Confirm that both the DEPT and EMP tables are stored in the data dictionary. (*Hint:* USER_TABLES)

-
  ```
  select table_name from user_tables where table_name in('DEPT1' , 'EMP1');
  ```

5. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY, and DEPARTMENT_ID columns. Name the columns in your new table ID, FIRST_NAME, LAST_NAME, SALARY , and DEPT_ID, respectively.

-
  ```
  create table EMPLOYEESS2 as select emp_id,emp_name,emp_last_name,salary from EMP;
  ```

-
  ```
  desc EMPLOYEESS2;
  ```

-
  ```
  alter table EMPLOYEESS2 add(DEPARTMENT_ID number);
  ```

- alter table EMPLOYEESS2 rename column emp_id to id;
- alter table EMPLOYEESS2 rename column emp_name to first_name;
- alter table EMPLOYEESS2 rename column emp_last_name to last_name;

- desc EMPLOYEESS2;

6. Drop the EMP table.

- drop table EMP;

7. Rename the EMPLOYEES2 table as EMP.

- alter table EMPLOYEESS2 rename to EMP;

- desc EMP;

8. Add a comment to the DEPT and EMP table definitions describing the tables. Confirm your additions in the data dictionary.

- COMMENT ON TABLE EMP IS 'Employees Data';

- SELECT * FROM USER_TAB_COMMENTS where table_name='EMP';

- COMMENT ON TABLE DEPT IS 'Departmenr Data';

- SELECT * FROM USER_TAB_COMMENTS where table_name='DEPT';

9. Drop the FIRST_NAME column from the EMP table. Confirm your modification by checking the description of the table.

- alter table EMP drop column first_name;

- desc EMP

10. In the EMP table, mark the DEPT_ID column in the EMP table as UNUSED. Confirm your modification by checking the description of the table.

- alter table EMP set unused(DEPARTMENT_ID);

- desc EMP;

11. Drop all the UNUSED columns from the EMP table. Confirm your modification by checking the description of the table

- alter table EMP drop unused column;

- desc EMP;

Part 2: Including Constraints

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

- **Describe constraints**
- **Create and maintain constraints**

<u>What are Constraints?</u>

- **Constraints enforce rules at the table level.**
- **Constraints prevent the deletion of a table if there are dependencies.**
- **The following constraint types are valid:**
  - **NOT NULL**
  - **UNIQUE**
  - **PRIMARY KEY**
  - **FOREIGN KEY**
  - **CHECK**

<u>The NOT NULL Constraint</u>

```
create table emp3(emp_name varchar2(10), department_name varchar2(10), emp_id number not null);
```

<u>The UNIQUE Constraint</u>

```
create table emp5(emp_name varchar2(10) not null, department_name varchar(10), emp_id number UNIQUE);
```

<u>The PRIMARY KEY Constraint</u>

```
create table emp6(emp_name varchar2(10) not null, department_name varchar(10), emp_id number primary key);
```

<u>The FOREIGN KEY Constraint</u>

```
create table emp7(emp_name varchar2(10) not null, department_name varchar(10), emp_id, foreign
key(emp_id) references emp6(emp_id));
```

The CHECK Constraint

```
create table emp1(emp_name varchar2(20),emp_id number,salary number,emp_branch varchar(20),check
(emp_id>5));
```

```
desc emp1;
```

Dropping a Constraint

```
alter table LDRP6 drop Constraint my_ldrp_id_pk (primary key name);
```

Disabling Constraints

```
alter table LDRP6 disable constraint my_ldrp_id_pk;
```

Enabling Constraints

```
alter table LDRP6 enable Constraint my_ldrp_id_pk;
```

Cascading Constraints

```
Create table ldrp6(sub varchar(10) primary key,state varchar(10), id int , foreign key(id) references
ldrp(id) On delete set null);

Desc ldrp6;
```

Viewing Constraints

```
Select * from user_constraints where table_name = ‘ldrp6’;
```

Exercises

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.
Hint: The constraint is enabled as soon as the ALTER TABLE command executes successfully.

```
alter table EMP add constraint my_emp_id_pk primary key (id);


desc EMP;
```

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.
Hint: The constraint is enabled as soon as the ALTER TABLE command executes successfully.

```
alter table DEPT add constraint my_dept_id_pk primary key (dept_id);


desc DEPT;
```

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

```
alter table EMP add(DEPT_ID number);
```

```
desc EMP;

alter table EMP add constraint my_emp_dept_id_fk foreign key(DEPT_ID) references DEPT(dept_id);

desc EMP;
```

4. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called lab10_4.sql.

```
select constraint_name , constraint_type from user_constraints where table_name in ('EMP3' ,
'DEPT2');
```

5. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.

```
select object_name , object_type from user_objects where object_name like 'DEPT2%';

select object_name , object_type from user_objects where object_name like 'EMP3%';
```

6. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale

```
alter table EMP add( COMMISSION number(2,2));

desc EMP;
```

7. Add a constraint to the commission column that ensures that a commission value is greater than zero.

```
alter table EMP modify( COMMISSION number(2,2) check (COMMISSION>0));

desc EMP;
```

Date:-		**Sign:-**		**Grade:-**	

Practical 2: Manipulating Data with Database Objects (DML)

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

- **Describe each DML statement**
- **Insert rows into a table**
- **Update rows in a table**
- **Delete rows from a table**
- **Merge rows in a table**
- **Control transactions** ▪

Inserting New Rows

```
insert into empl(emp_name,emp_id,dept_id) values('jay',113,05);
```

Inserting Rows with Null Values

```
insert into empl(emp_name,emp_id,dept_id) values('Amit',null,null);
```

Creating a Script

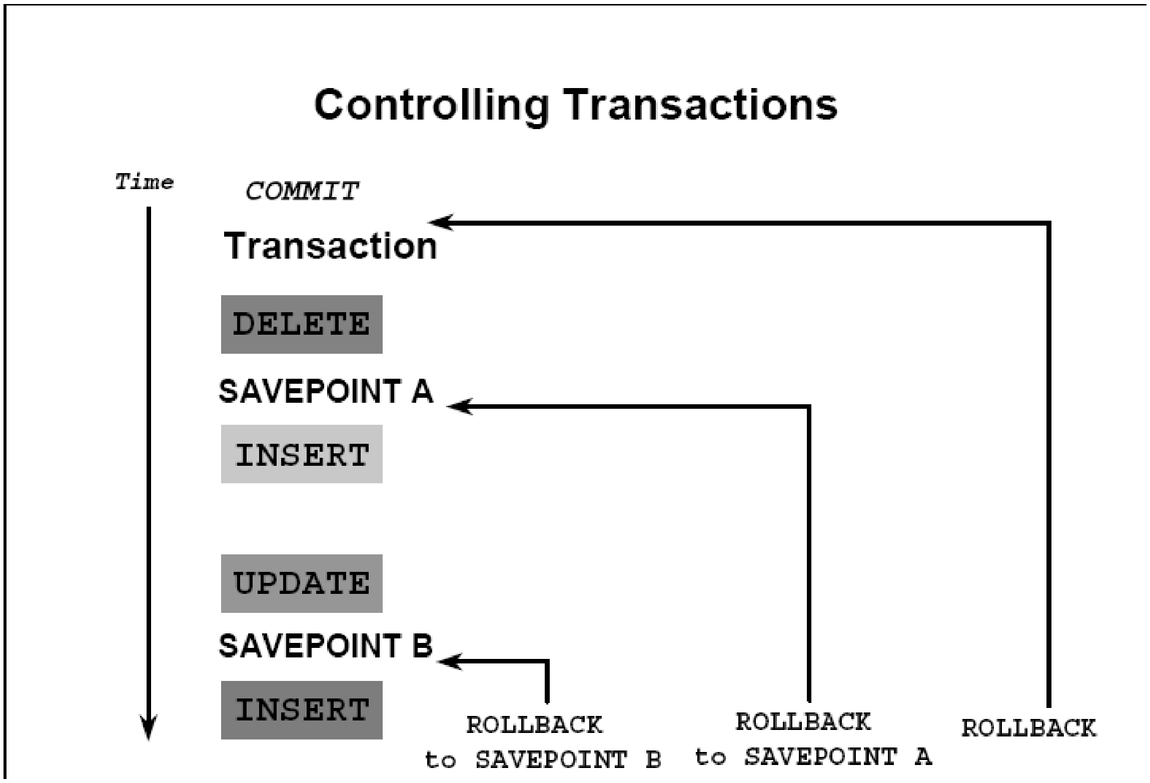
```
INSERT INTO departments (department_id , department_name , location_id) VALUES (&department_id
, „&department_name‟ , &location_id);
```

Updating Rows in a Table

```
update empl set dept_id=0 where emp_id=113
```

ELETE Statement

elete from empl where emp_name='jay';



Controlling Transactions

Exercises

1. Describe the structure of the EMPLOYEES table to identify the column names.

```
create table MY_EMPLOYEE(id number(4) constraint my_employee_id_nn not null,last_name
varchar2(25),first_name varchar2(25),userid varchar2(25),salary number(9,2));

  desc MY_EMPLOYEE;
```

2. Add the first row of data to the EMPLOYEES table from the following sample data. Do not list the columns in the INSERT clause.

```
insert into MY_EMPLOYEE values(1,'patel','deep','dpprr',200000);
```

3. Populate the EMPLOYEEStable with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
insert into MY_EMPLOYEE(id,last_name,first_name,userid,salary)
values(2,'brrott','dharmesh','bino',6578);
```

4. Confirm your addition to the table.

```
select * from MY_EMPLOYEE;
```

5. Write an insert statement in a text file named loademp.sql to load rows into the EMPLOYEEStable. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID.

```
set echo off set verify off insert into EMP5 values(&p_id , '&p_last_name' , '&p_first_name' ,
lower(substr('&p_first_name',1,1) || substr('&p_last_name'1,7)) , &p_salary);
```

```
set verify on
set echo on
```

6. Populate the table with the next two rows of sample data by running the insert statement in the script that you created.

```
set echo off set verify off insert into EMP5 values(&p_id , '&p_last_name' , '&p_first_name' ,
lower(substr('&p_first_name',1,1) || substr('&p_last_name'1,7)) , &p_salary);
set verify on
set echo on
```

7. Confirm your additions to the table.

```
select * from MY_EMPLOYEE;
```

8. Make the data additions permanent.

```
commit;
```

9. Update and delete data in the EMPLOYEES table.

10. Change the last name of employee 3 to Drexler

```
update MY_EMPLOYEE set last_name='DRExler' where id=3;
```

11. Change the salary to 1000 for all employees with a salary less than 900.

```
update MY_EMPLOYEE set salary=1000 where salary<900;
```

12. Verify your changes to the table.

```
select last_name,salary from MY_EMPLOYEE;
```

13. Delete Betty Dancs from the EMPLOYEEStable.

```
delete from MY_EMPLOYEE where last_name='brrott';
```

14. Confirm your changes to the table.

```
delete from MY_EMPLOYEE where last_name='brrott';
```

15. Commit all pending changes.

```
Commit;
```

16. Control data transaction to the EMPLOYEEStable.

```
SELECT DISTINCT job_id FROM employees;.
```

17. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

```
set echo off set verify off insert into EMP5 values(&p_id , '&p_last_name' , '&p_first_name' ,
lower(substr('&p_first_name',1,1) || substr('&p_last_name'1,7)) , &p_salary);
set verify on

set echo on
```

18. Confirm your addition to the table.

```
select * from MY_EMPLOYEE;
```

19. Mark an intermediate point in the processing of the transaction.

```
SQL>savepoint step_18;
        Savepoint created.
```

20. Empty the entire table.

```
delete from EMP5;
```

21. Confirm that the table is empty.

```
select * from EMP5;
```

22. Discard the most recent DELETE operation without discarding the earlier INSERT operation.

```
SQL>rollback; Rollback complete.
```

23. Confirm that the new row is still intact.

```
select * from EMP5;
```

24. Make the data addition permanent.

```
Commit;
```

Date:-		**Sign:-**		**Grade:-**	

cal 3: Retrieving, Restricting and Sorting Data (DRL)

Objectives

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

- **List the capabilities of SQL SELECTstatements**
- **Execute a basic SELECTstatement**

Basic SELECTStatement

```
select * from empq;
```

Selecting Specific Columns

```
select emp_name,emp_id from empq;
```

Using Arithmetic Operators

```
select emp_id+dept_id from empq;

select emp_id*dept_id from empq;
```

Using the Concatenation Operator

```
select emp_name || 'has dept_id is' || dept_id from empq;
```

Eliminating Duplicate Rows

```
select distinct * from empq;
```

Displaying Table Structure

```
desc EMP6;
```

Exercise

Initiate an SQL*Plus session using the user ID and password provided by the instructor.

□ DONE

2. The following SELECTstatement executes successfully: SELECT last_name, job_id, salary AS Sal FROM employees;

Ans: [✓ True] False

3. The following SELECTstatement executes successfully:

```
SELECT * FROM job_grades;
```

Ans: [✓ True] False

4. There are coding errors in this statement. Can you identify them?

```
SELECT    employee_id, last_name sal x 12
          ANNUAL SALARY
FROM      employees;
```

Ans:

1. The employees table does not contain a column called sal . the column is called SALARY.
2. The multiplication operator is* not x as shown .
3. A comma is missing after the column last_name

5. Show the structure of the DEPARTMENTS table. Select all data from the table. Ans:

```
desc departments;

select * from departments;
```

Show the structure of the EMPLOYEEStable. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab1_7.sql.

```
desc EMPLOYEES1;
```

```
select emp_id,last_name,job_id,hire_date from EMPLOYEES1;
```

7. Run your query in the file lab1_7.sql.

8. Create a query to display unique job codes from the EMPLOYEEStable.

```
select distinct job_id from EMPLOYEES1;
```

9. Copy the statement from lab1_7.sqlinto the *i*SQL*Plus Edit window. Name the column headings Emp#, Employee, Job, and HireDate, respectively. Run your query again.

```
select emp_id "EMP#",last_name "EMPLOYEE#",job_id "JOB#",hire_date "HIRED
DATE#" from EMPLOYEES1;
```

10. Display the last name concatenated with the job ID, separated by a comma and space, and name the column EmployeeandTitle.

```
select last_name || ',' ||job_id "EMPLOYEE AND TITLE" from EMPLOYEES1;
```

11. Create a query to display all the data from the EMPLOYEEStable. Separate each column by a comma. Name the column THE_OUTPUT.

```
select employee_id || ' , ' || first_name || ' , ' || last_name || ' , ' || email || ' , ' ||
phone_number || ' , ' || job_id || ' , ' || manager_id || ',' || hired_date ||' '|| salary
The_OUTPUT from employee;
```

: Restricting and Sorting Data

Limiting Rows Using a Selection

- select last_name,salary,first_name from emp1 limit 1;

Character Strings and Dates

- select convert(datetime , ‘20190731’) from emp1;

Comparison operators

- select first_name,last_name from emp1 where salary=50000;

Other Comparison Conditions

- select first_name,last_name from emp1 where salary>=50000;
- select first_name,last_name from emp1 where salary<50000;

Logical Conditions

- select first_name,last_name from emp1 where salary>70000 OR salary<60000 ;

ORDER BY Clause

- select * from emp1 order by salary desc;

Exercise

1. Create a query to display the last name and salary of employees earning more than $12,000. Place your SQL statement in a text file named lab2_1.sql. Run your query.

```
select last_name , salary from EMPLOYEES12 where salary>12000;
```

2. Create a query to display the employee last name and department number for

yee number 176.

```
select last_name , job_id from EMPLOYEES12 where emp_id = 1;
```

3. Modify lab2_1.sql to display the last name and salary for all employees whose salary is not in the range of $5,000 and $12,000. Place your SQL statement in a text file named lab2_3.sql.

```
select last_name , salary from EMPLOYEES12 where salary not between 5000 AND 12000;
```

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998. Order the query in ascending order by start date.

```
select last_name , job_id ,hire_date from EMPLOYEES12 where hire_date between DATE
'2020-07-19' and DATE '2020-09-10' order by hire_date;
```

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

```
select last_name ,job_id from EMPLOYEES12 where job_id in(30,50) order by last_name
```

6. Modify lab2_3.sql to list the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Resave lab2_3.sql as lab2_6.sql. Run the statement in lab2_6.sql.

```
select last_name "Employee" , salary "monthly salary" from EMPLOYEES12 where salary
between 5000 AND 12000 AND job_id in(20,50);
```

7. Display the last name and hire date of every employee who was hired in 1994.

```
select last_name , hire_date from EMPLOYEES12 where hire_date like '%20';
```

8. Display the last name and job title of all employees who do not have a manager.

```
select last_name , job_id from EMPLOYEES12 where manager_id is NULL;
```

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

```
select last_name , salary , commission_pct from EMPLOYEES12 where commission_pct is not NULL order by salary desc , commission_pct desc;
```

10. Display the last names of all employees where the third letter of the name is an *a.*

```
select last_name from EMPLOYEES12 where last_name like ' a%';
```

11. Display the last name of all employees who have an *a* and an *e* in their last name.

```
select last_name from EMPLOYEES12 where last_name like '%a%' and last_name like '%e%';
```

12. Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to $2,500, $3,500, or $7,000.

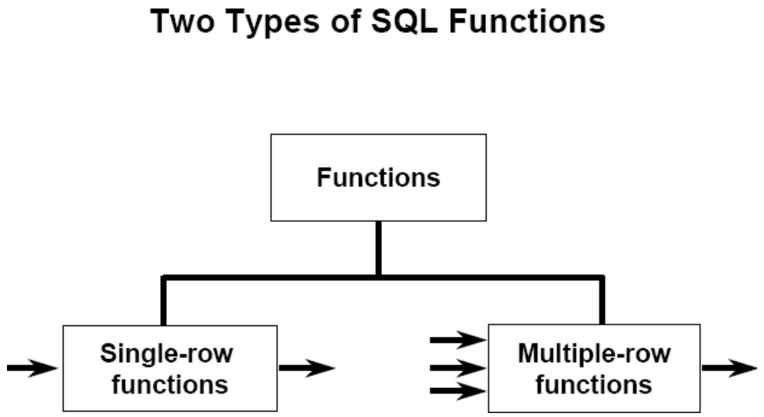
```
Select last_name , job_id , salary from EMPLOYEES12 where job_id in(‘sa_rep’ ,’st_clerk’)
AND salary not in (2500 ,3500, 7000);
```

13. Modify lab2_6.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab2_6.sql as lab2_13.sql. Rerun the statement in lab2_13.sql.

```
select last_name "EMPLOYEE" , salary "Monthly Salary" , commission_pct from
EMPLOYEES12 where commission_pct = 800;
```

Date:-		**Sign:-**		**Grade:-**	

al 4: SQL Single Row Functions


Two Types of SQL Functions

Character Functions

- select upper(last_name) from empl1;
- select initcap(last_name) from empl1;
- select lower(last_name) from empl1;

Case Manipulation Functions

- select length(first_name), length(last_name) from empl1;
- select concat(first_name,last_name) from empl1;
- select INSTR('last_name', 't') from empl1;
- select trim('e' from 'last_name') from empl1;

Character-Manipulation Functions

Number Functions

- select round(45.2564, 2) from empl1;

NVL Function

- select last_name, salary ,NVL(commision,0),(salary*12),(salary*12+NVL(commision,0)) as "anl_sal" from empl1;

the NVL2 Function

- select last_name, salary, commision, NVL2(commision, 'sal+comm', 'sal')income from empl1;

Using the NULLIF Function

- Select first_name,length(first_name) “expr 1”,last_name,length(last_name) “expr 2”, NULLIF(length(first_name),length(last_name)) result from empl1;

Using the COALESCE Function

- Select COALESCE(NULL, ‘Jainam’ , ‘NULL’, ‘Kothari’);

Using the CASE Expression

```
Select last_name,salary CASE
     When salary>=1200 then ‘salary is more than 12000’ Else
     ‘salary is less than 12000’
     From empl1;
```

Using the DECODE Function

Select last_name,DECODE(emp_id,1001, ‘Employee1’, 1002, ‘Employee2’ , ‘Employee other than 1 and 2’)from empl1;

Exercise

1. Write a query to display the current date. Label the column Date.

- select dt "Date" from empl1;

2. For each employee, display the employee number, last_name, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary. Place your SQL statement in a text file named lab3_2.sql.

- select emp_id, last_name, salary, salary*1.5 + salary as "new salary" from empl2;

5. Modify your query lab3_2.sql to add a column that subtracts the old salary from the new salary. Label the column Increase. Save the contents of the file as lab3_4.sql. Run the revised query.

-
  ```
  select emp_id, last_name, salary, salary*1.5 + salary as "new salary",salary*1.5 - salary as "Increse" from empl2;
  ```

4. Write a query that displays the employee’s last names with the first letter capitalized and all other letters lowercase, and the length of the names, for all employees whose name starts with J, A, or M. Give each column an appropriate label. Sort the results by the employees’ last names.

- select initcap(last_name) "NAME",length(last_name) "LENGTH" from empl2 where last_name like 'p%';

5. For each employee, display the employee’s last name, and calculate the number of months between today and the date the employee was hired. Label the column ONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

- select last_name, round(months_between (SYSDATE, dt)) months_worked from empl1 order by months_between(SYSDATE, dt);

6. Write a query that produces the following for each employee:
<employee last name> earns <salary> monthly but wants <3 times salary>. Label the column Dream Salaries.

- select last_name||'earns' || salary || 'monthly but wants '||salary*3||'as dream salary' from empl2;

7. Create a query that displays the employees’ last names and commission amounts. If an employee does not earn commission, put “No Commission.” Label the column COMM.

- select last_name,first_name,NVL(TO_CHAR(commision),'NO Commision') COMM from empl2;

8. Create a query that displays the employees’ last names and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.

- select rpad(first_name,8)||"||rpad(",salary/1000+1,'*') EMPLOYEES_AND_THEIR_SALARIES from empl2 order by salary;

A PRACTICAL

❖ **Create table for student with Roll Number , First name , Last name , Branch , Sem , Hobby , Date of birth.**

 create table Student(roll_no number (10) , first_name varchar2(20) , last_name varchar2(20) , branch varchar2(30) , Sem number(5) , hobby varchar2(50) , d_o_b DATE);

1) **convert first and last name into upper case.**

 select upper(first_name) , upper(last_name) from Student;

2) **Convert Hobby into Lower Case.**

 select lower(hobby) from Student;

3) **Display date of birth of Every student with Name.**

 select d_o_b , first_name || ' ' || last_name FULLNAME from Student;

4) **Show system date(sysdate).**

 select sysdate "DATE" from Student;

5) **Display the Length of First_name and Last_name.**

- select length(first_name) , length(last_name) from Student;

) **Show the use of INITCAP.**

- select initcap(first_name) ||' ' || initcap(last_name) FULL_NAME from Student;

7) **Make roll number as primary key.**

- alter table Student add constraint roll_no_pk primary key(roll_no);

8) **Make semester Number as not null.**

- alter table Student modify Sem number(4) not null;

9) **Add(concate) both branch and semester.**

- select branch ||' '|| Sem BRANCH_AND_SEM from Student;

LDRP Institute of Technology and Research

Date:-		**Sign:-**		**Grade:-**	

cal 5: SQL Multiple Row Functions (Aggregate Functions)

Using the AVG and SUM Functions

- select sum(salary) from empd;
- select avg(salary) from empd;

Using the MIN and MAX Functions

- select MIN(salary) from empd;
- select MAX(salary) from empd;

Using the COUNT Function

- select count(commission) from empd;

Group Functions and Null Values

- select last_name , count(*) as "Num of employees" , avg(salary) as "Avg. Dept. Salary" from empd group by last_name order by last_name NULLS LAST;
- select last_name , count(*) as "Num of employees" , sum(salary) as "sum Dept. Salary" , avg(salary) as "Avg. Dept. Salary" from empd group by last_name order by last_name NULLS LAST;

Using the GROUP BY Clause

- select sum(salary) , max(commission) , min(emp_id) from empd group by salary;

Using the GROUP BY Clause on Multiple Columns

- select sum(salary) , max(commission) , min(emp_id) from empd group by (salary , commission);

Excluding Group Results: The HAVING Clause

- select sum(salary) , max(commission) , min(emp_id) , min(last_name) from empd group by (salary) having salary>50000;

- select sum(salary) , max(commission) , min(emp_id) , min(last_name) from empd group by (commission , salary) having (commission > 10) AND (salary = 50000);

- select sum(salary) , max(commission) , min(emp_id) , min(last_name) from empd group by (commission) having commission>1;

Exercises

Determine the validity of the following three statements.

1. Group functions work across many rows to produce one result per group.

   True False

2. Group functions include nulls in calculations.

   True **False**

3. The WHERE clause restricts rows prior to inclusion in a group calculation.

   True False

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number.

   - select emp_id , ROUND(max(salary),0) "Maximum" , ROUND(min(salary),0) "Minimun" , ROUND(SUM(salary),0) "Sum" , ROUND(AVG(salary),0) "Average" from empd group by emp_id;

Modify the query in exe_4.sql to display the minimum, maximum, sum, and average salary for each job type.

- select emp_id, count(*) from empd group by emp_id;

6. Write a query to display the number of people with the same job.

7. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER_ID column to determine the number of managers.*

- select count(emp_id) "Number of Managers" from empd;

8. Write a query that displays the difference between the highest and lowest salaries. Label the column DIFFERENCE.

- select max(salary)-min(salary) as "DIFFERENCE" from empd;

9. Display the manager number and the salary of the lowest paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

- select emp_id , min(salary) from empd where emp_id IS NOT NULL group by emp_id having Min(salary) > 6000 order by min(salary) desc;

Date:-		**Sign:-**		**Grade:-**	

cal 6: Displaying Data from Multiple Tables (Join)

Cartesian Products

- select last_name , dept_name from empd , deptll;

Retrieving Records with Equijoins

- select empd.last_name , empd.emp_id , empd.first_name , deptll.dept_id , deptll.dept_name from empd , deptll;

- select empd.last_name , empd.emp_id , empd.first_name , deptll.dept_id , deptll.dept_name from empd , deptll where empd.emp_id = deptll.dept_id;

Using Table Aliases

- select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name from empd e , deptll d;

- select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name from empd e, deptll d where e.emp_id = d.dept_id;

Joining More than Two Tables

- select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name , l.city , l.location_id from empd e , deptll d , location l where e.emp_id = d.dept_id AND d.dept_id = l.location_id;

Retrieving Records with Non-Equijoins

- select empd.last_name , empd.emp_id , empd.first_name , deptll.dept_id , deptll.dept_name from empd , deptll where empd.emp_id != deptll.dept_id;

- select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name from empd e, deptll d where e.emp_id != d.dept_id;

Using Outer Joins

- select w.last_name , e.emp_id from empd w , empd e;

Joining a Table to Itself (Self Join)

- select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name from empd e , deptll d where e.emp_id (+) = d.dept_id;

Exercises

1. Write a query to display the last name, department number, and department name for all employees.

```
select last_name , emp_id , first_name from empd e ,deptll d;
```

2. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

```
select distinct job_id , location_id from empd , deptll where empd.emp_id = deptll.dept_id AND empd.emp_id = 21;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

```
select e.last_name , d.dept_name , d.location_id from empd e, deptll d where e.emp_id = d.dept_id AND e.commission IS NOT NULL;
```

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named `lab4_4.sql`.

```
select last_name , dept_name from empd , deptll where empd.emp_id = deptll.dept_id AND last_name like '%a%';
```

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

```
select e.last_name , e.emp_id , d.dept_name from empd e JOIN deptll d ON (e.emp_id = d.dept_id) where upper(d.desig) = 'manager';
```

6. Display the employee last name and employee number along with their manager’s last name and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively.

```
select e.last_name "employee" , e.emp_id "EMP#" , d.desig "Manager" , e.emp_id "Mgr#" from empd e JOIN
deptll d ON (d.dept_id = e.emp_id);
```

7. Create a query that displays employee last names, department numbers, and all the employees

who work in the same department as a given employee. Give each column an appropriate label.

```
select e.last_name , e.emp_id , e.first_name , d.dept_id , d.dept_name , l.city , l.location_id from empd e , deptll d
     , location l where e.emp_id = d.dept_id AND d.dept_id = l.location_id;
```

8. Show the structure of the `JOB_GRADES` table. Create a query that displays the name, job, department name, salary, and grade for all employees.

Date:-		**Sign:-**		**Grade:-**	

Practical 7: Using Commit and Rollback show Transaction ACID Property.

- create table IPL(playre_name varchar2(20) , player_no number , player_team varchar(20) , player_country varchar2(20) , player_runs number , player_bestscore number);

- desc IPL;

- insert into IPL values('Rohit' , 11 , 'MI' , 'INDIA' , 50 , 300);
- insert into IPL values('Mahendra' , 1 , 'CSK' , 'INDIA' , 70 , 250);
- insert into IPL values('Rahul' , 12 , 'KXII' , 'INDIA' , 40 , 150);
- insert into IPL values('Shikhr' , 10 , 'DD' , 'INDIA' , 80 , 210);
- insert into IPL values('Rishbh' , 8 , 'RR' , 'INDIA' , 60 , 200);

- select * from IPL;

- commit;

insert into IPL values('Polard' , 18 , 'MI' , 'WESTINDIS' , 66 , 260);

insert into IPL values('Smith' , 23 , 'RR' , 'AUSTRALIYA' , 55 , 220);

rollback;

savepoint

```
insert into IPL values('Gayel' , 28 , 'KXII' , 'WESTINDIS' , 100 , 460);

insert into IPL values('Bravo' , 5 , 'CSK' , 'WESTINDIS' , 40 , 160);


savepoint dbms_1;


insert into IPL values('Jasprit' , 40 , 'MI' , 'INDIA' , 44 , 140);


savepoint dbms_2;
```

```
insert into IPL values('Virat' , 20 , 'RCB' , 'INDIA' , 90 , 300);


rollback dbms_1;
```

Date:-		**Sign:-**		**Grade:-**	

tical 8: Securing data using Views and Controlling User Access (DCL)

Part 1: Creating Views

Simple Views and Complex Views

- create view det_vu as select desig,location from employee;(Simple View)

- create view dept vu̅ as select desig,location from employee wheree_id=24; (Complex View)

Creating a View

- create view dept vu̅ as select desig,location from employee;

Retrieving Data from a View

- select * from dept vu̅;

Modifying a View

- create or replace view dept_vu as select e_id,e_name,desig,location from employee;

  select * from dept_vu;

Creating a Complex View

- create view dept_vu1 as select e_id,e_name,desig,location from employee where

dep_id=30; select * from dept_vu1;

Rules for Performing DML Operations on a View

```
create or replace view InsertEmp (e_id,e_name,dep_id,dept_name,location_id,location,job_id,desig) as
select * from employee;

    Insert into InsertEmp(e_id,e_name,dep_id,dept_name,location_id,location,job_id,desig) values(50,
    'John',70,'Administration',001,'Ahmedabad',10006,'Manager');
```

elect * from InsertEmp;

Rules for Performing DML Operations on a View

- update InsertEmp Set location = 'MUMBAI' where dep_id = 70;

elect * from InsertEmp;

'or Performing DML Operations on a View

- delete From InsertEmp where dep_id = 70;

  select * from InsertEmp;

Removing a View

- drop view dept_vu1;

  select * from dept_vu1;

Top-N Analysis

Exercises

1. Create a view called EMPLOYEES_VUbased on the employee numbers, employee names, and department numbers from the EMPLOYEEStable. Change the heading for the employee name to EMPLOYEE.

- create view EMPLOYEES_VU as select e_id,e_name as "EMPLOYEES",dep_id from employee;

2. Display the contents of the EMPLOYEES_VUview.

- select * from EMPLOYEES_VU;

3. Select the view name and text from the USER_VIEWSdata dictionary view.

   Note: Another view already exists. The EMP_DETAILS_VIEWwas created as part of your schema.

   Note: To see more contents of a LONGcolumn, use the *i*SQL*Plus command SET LONG n, where nis the value of the number of characters of the LONGcolumn that you want to see.

```
 set long 60 select view_name, text from user_views;
```

4. Using your EMPLOYEES_VUview, enter a query to display all employee names and department numbers.

```
 select employee, dep_id from employees_vu;
```

5. Create a view named DEPT50that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

```
 create view dept50 as select e_id as "EMPNO", e_name as
"EMPLOYEE",dep_id as "DEPTNO" from employee where dep_id = 30 with
check option CONSTRAINT emp_dept_50;
```

6. Display the structure and contents of the DEPT50view.

```
 desc dept50;
select * from dept50;
```

7. Attempt to reassign Matos to department 0.

```
 update dept50 set deptno =0 where e_name = ‘Matos’;
```

8. Create a view called SALARY_VUbased on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES, DEPARTMENTS, and JOB_GRADEStables. Label the columns Employee, Department, Salary, and Grade, respectively.

```
 create or replace view SALARY_VU as select e.e_name as "Employee",d.dept_name as
"Department",j.salary as "Salary",j.salary_grade as "Salary Grades" from employee e,department
d,job_grades j where e.e_id=d.e_id AND j.salary between j.lowest_sal and j.highest_sal;
```

```
select * from SALARY_VU;
```

: Privileges

• Database security:

– System security

– Data security

- **System privileges: Gaining access to the database**
- **Object privileges: Manipulating the content of the Database objects**
- **Schemas: Collections of objects, such as tables, Views, and sequences**

Object Privileges

• Object privileges vary from object to object.

• An owner has all the privileges on the object.

• An owner can give specific privileges on that owner's object. GRANT *object_priv* [(*columns*)]
ON *object*
TO {*user*|*role*|PUBLIC}
[WITH GRANT OPTION];

Grant query privileges on the EMPLOYEES table.

```
grant all on EMPLOYEES to public;
```

Grant privileges to update specific columns to users and roles

```
create user employee_admin identified by Password;
grant update on EMPLOYEES to employee_admin;

create user employee identified by Password;
grant update(employee_name) on EMPLOYEES to employee;
```

Give a user authority to pass along privileges.

```
create user admin identified by Password;
grant select,insert on EMPLOYEES to admin with grant option;
```

As user revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

```
evoke select,insert on EMPLOYEES from admin;
```

Exercise:

1. Grant another user access to your DEPARTMENTS table. Have the user grant you query access to his or her DEPARTMENTS table.

   - create user u1 identified by team1;

     create user u2 identified by team2; grant
     select on DEPARTMENTS to u1; grant
     select on DEPARTMENTS to u2;

2. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources department number 510. Query the other team's table.

   - insert into DEPARTMENTS(dept_id, dept_name) values(500, 'Education');
   - insert into DEPARTMENTS(dept_id,dept_name)values(510, 'Administration');

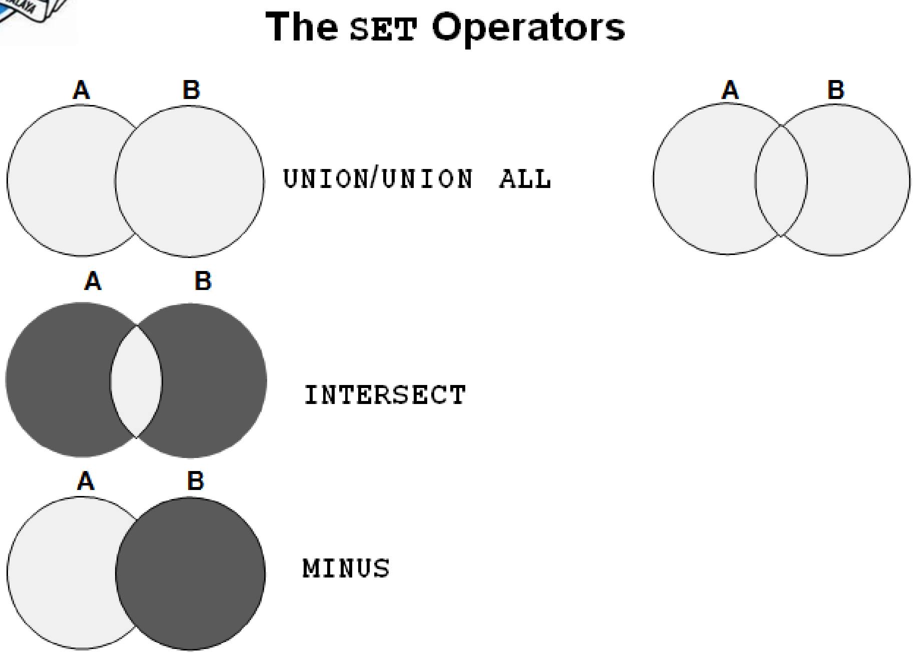
3. Revoke the SELECT privilege on your table from the other team

   - revoke select on DEPARTMENTS from u2;
   - revoke select on DEPARTMENTS from u1;

LDRP Institute of Technology and Research

Date:-		**Sign:-**		**Grade:-**	

:al 9: Database SET Operations



Using the UNIONOperator

select emp_name from employees union select dept_name from departments order by emp_name;

select 'emp_name' as type ,emp_id,emp_add from employees union select 'departments',dept_id,dept_add from departments;

Using the UNION ALLOperator

select emp_name from employees union all select dept_name from departments order by emp_name;

Using the INTERSECTOperator

- from employees intersect select dept_add from departments;

The MINUSOperator

- select emp_add from employees minus select dept_add from departments;

Exercises

ist the department IDs for departments that do not contain the job ID ST_CLERK, using ETOperators.

-
```
select dept_id from departments minus select dept_id from employees where
job_id='ST_CLERK';
```

2. Display the country ID and the name of the countries that have no departments located in them, using SEToperators.

-
```
select country_id,country_name from countries minus select
l.country_id, c.country_name from locations l,countries c where
l.country_id=c.country_id;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID, using SET operators.

```
select job_id,emp_id,dept_id,'x' dummy from employees where dept_id=10 union

select job_id,emp_id,dept_id,'y' from employees where dept_id=50 union

select job_id,emp_id,dept_id ,'z' from employees where dept_id=20 order by 3;
```

4. List the employee IDs and job IDs of those employees who currently hold the job title that they held before beginning their tenure with the company.

```
select emp_id,job_id from employees intersect select emp_id,job_id from job;
```

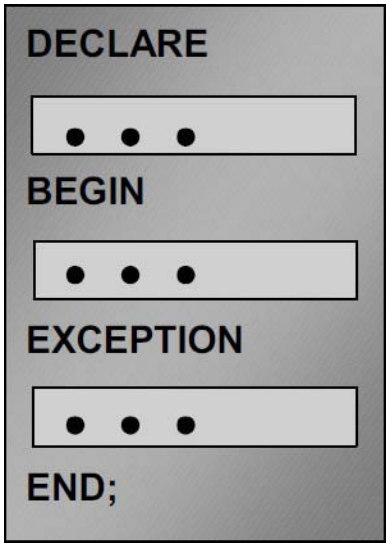
5. Write a compound query that lists the following:
   - Last names and department ID of all the employees from the EMPLOYEEStable, regardless of whether or not they belong to any department or not
   - Department ID and department name of all the departments from the DEPARTMENTStable, regardless of whether or not they have employees working in them

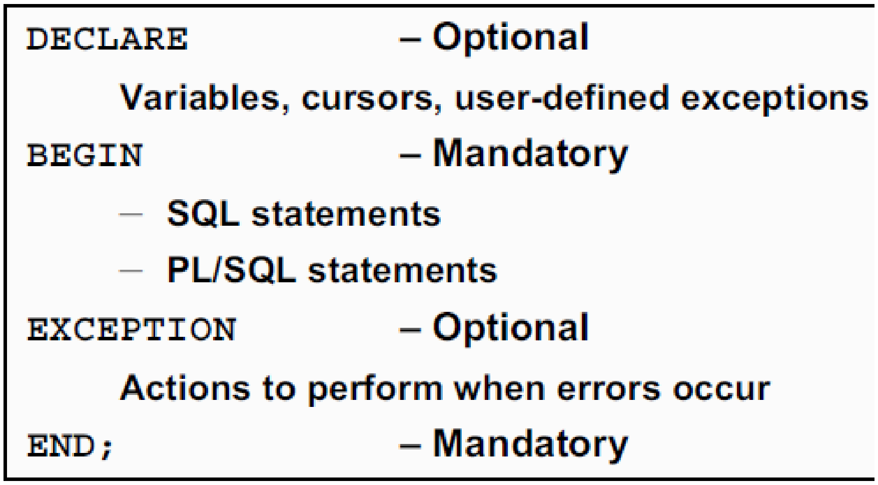
```
select emp_name,dept_id,to_char(null) from employees union select
to_char(null),dept_id,dept_name from departments;
```

Date:-		**Sign:-**		**Grade:-**	

Practical 10: PL/SQL Block Syntax and DML Operation through PL/SQL Block

PL/SQL Block:





```
DECLARE              – Optional
        Variables, cursors, user-defined exceptions
BEGIN                – Mandatory
        – SQL statements
        – PL/SQL statements
EXCEPTION            – Optional
        Actions to perform when errors occur
END;                 – Mandatory
```

Executing PL/SQL Statements and Block

```
BEGIN

dbms_output.put_line ('Hello World!');

dbms_output.put_line ('I am Jainam Kothari');

END;
```

Declaring variable with %TYPE Attribute

```
    DECLARE
emp_name employees.name_id%TYPE;
    BEGIN
select emp_name into emp_name from employees where emp_id=154;
dbms_output.put_line (emp_name);
    END;
```

Using Bind Variables

```
EGIN
  o  update employees
  o  set emp_id=100 where dept_id=50;
END;
```

Using DBMS_OUTPU.PUT_LINE

BEGIN

- dbms_output.put_line ('Hello World!');
- dbms_output.put_line ('I am Jainam Kothari');

END;

SELECT Statement in PL/SQL

DECLARE

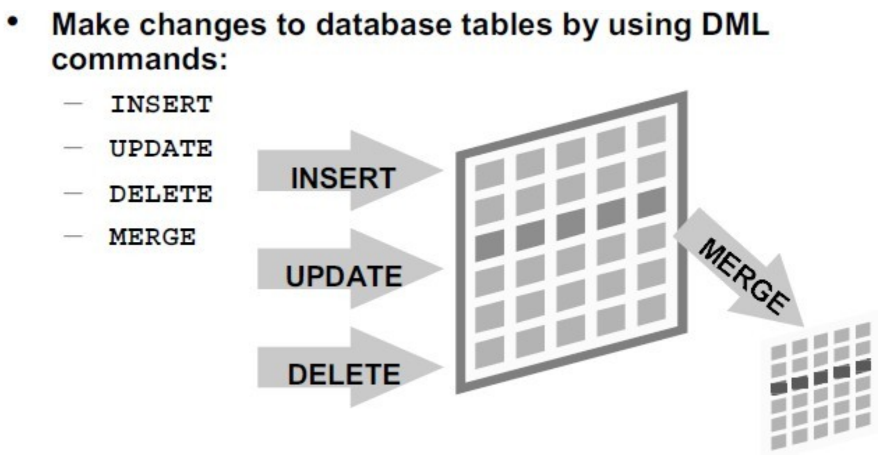
- emp_name employees.emp_name%TYPE;

BEGIN

- select emp_name into emp_name from employees where emp_id = 4871;
- dbms_output.put_line (emp_name);

END;

Manipulating Data using PL/SQL



- Make changes to database tables by using DML commands:
  - INSERT
  - UPDATE
  - DELETE
  - MERGE

se:

Add new Employee information into EMPLOYEES Table

- BEGIN
- insert into EMPLOYEES(emp_id,emp_name,emp_add) values(450,'ABC','Ahmedabad');
- END;
- select * from employees;

2. Increase Salary of of all employees who are clerks

- DECLARE
- sal_increase EMPLOYEES.salary%TYPE := 800;
- BEGIN
- update EMPLOYEES set salary = salary + sal_increase where job_id = 'ST_CLERK';
- END;
- select * from EMPLOYEES;

3. Delete rows who are belongs to department number 20

- BEGIN
  - delete EMPLOYEES where dept_id=20;
- END;
- select * from EMPLOYEES;

LDRP Institute of Technology and Research

Date:-		**Sign:-**		**Grade:-**	

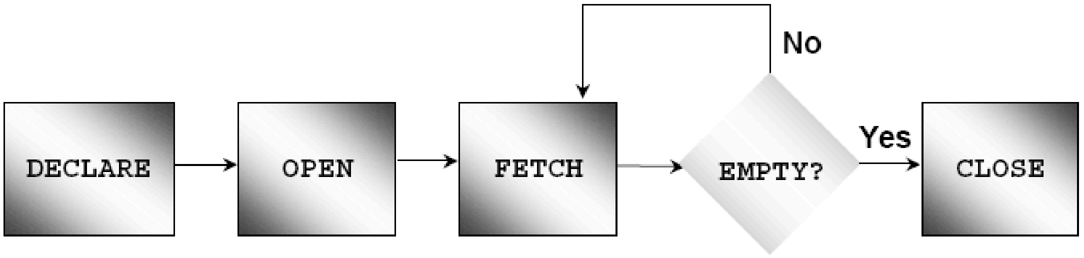
cal 11: Working with Cursor

ursors

SQL statement executed by the Oracle Server has an individual cursor associated with it:

- Implicit cursors: Declared for all DML and PL/SQL SELECT statements
- Explicit cursors: Declared and named by the programmer

Controlling Explicit Cursors



- Create a named SQL area
- Identify the active set
- Load the current row into variables
- Test for existing rows
- Return to FETCH if rows are found
- Release the active set

Declaring the Cursor

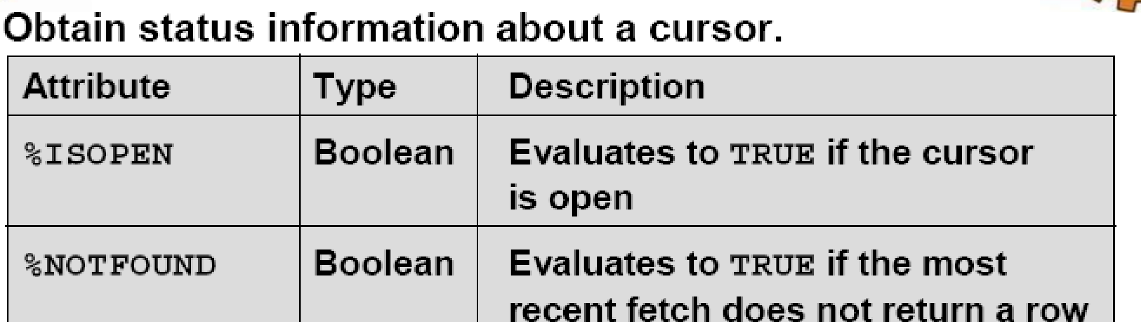
Explicit Cursor Attributes

Implicit Cursor Example:

WRITE A PL/SQL BLOCK TO CHANGE THE SALARY OF SPECIFIC EMPLOYEE. DISPLAY APPROPRIATE MESSAGES BASED ON THE RECORD FOUND OR NOT.

```
SET SERVERSTAUS ON
BEGIN
        UPDATE EMP SET SAL=&SALARY WHERE ENAME=&NAME;

        IF SQL%FOUND THEN
                DBMS_OUTPUT.PUT_LINE('SALARY IS CHANGED SUCC.');
        END IF;
                                              IF SQL%NOTFOUND
                                                      THEN
                                                      DBMS_OUTP
                                                      UT.PUT_LINE
                                      END;            ('NO RECORD
                                                      FOUND.');
                                              END IF;
```

Cursor Example:



Obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row

se:

ve the first five employees with job history

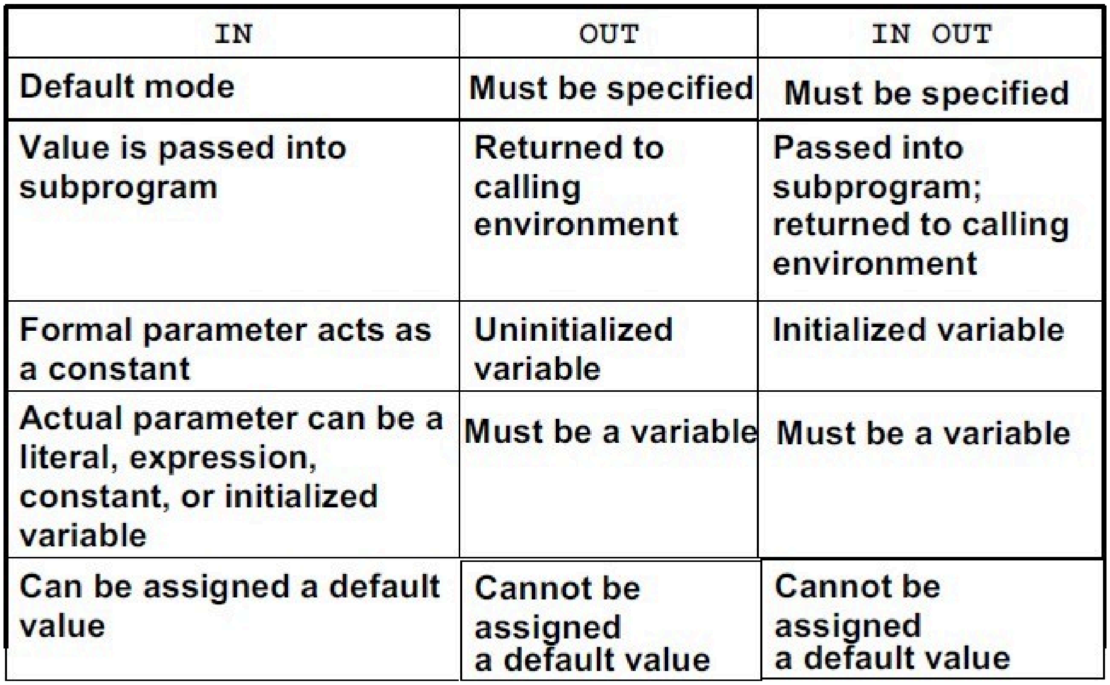
Date:-		**Sign:-**		**Grade:-**	

tical 12: Creating Procedures and Functions in PL/SQL

for creating Procedures

Formal versus Actual Parameters

Creating Procedure with Parameters



IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

ple of IN and OUT Parameter

Syntax for creating Functions

Creating and Executing Store Functions

ng Functions in SQL Expressions

Comparing Procedure and Function

Date:-		**Sign:-**		**Grade:-**	

tical 13: Creating Database Triggers

Types of Database Triggers

Syntax for creating DML Triggers

Example of Creating DML Statement Triggr

Date:-		**Sign:-**		**Grade:-**	

tical 14: Concept of Plan Table and Audit Trails

Using Plan Table:

The EXPLAIN PLAN statement displays execution plans chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. A statement's execution plan is the sequence of operations Oracle performs to run the statement.

The row source tree is the core of the execution plan. It shows the following information:

- An ordering of the tables referenced by the statement
- An access method for each table mentioned in the statement
- A join method for tables affected by join operations in the statement
- Data operations like filter, sort, or aggregation

Creating the PLAN_TABLE Output Table

Before issuing an EXPLAIN PLAN statement, you must have a table to hold its output. PLAN_TABLE is the default sample output table into which the EXPLAIN PLAN statement inserts rows describing execution plans. Use the SQL script UTLXPLAN.SQL to create the PLAN_TABLE in your schema.

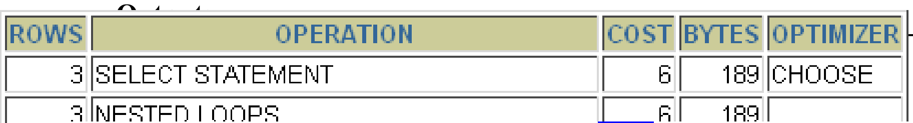
```
SQL> CONNECT HR/password
SQL> @$ORACLE_HOME/RDBMS/ADMIN/UTLXPLAN.SQL
```

Join Query:

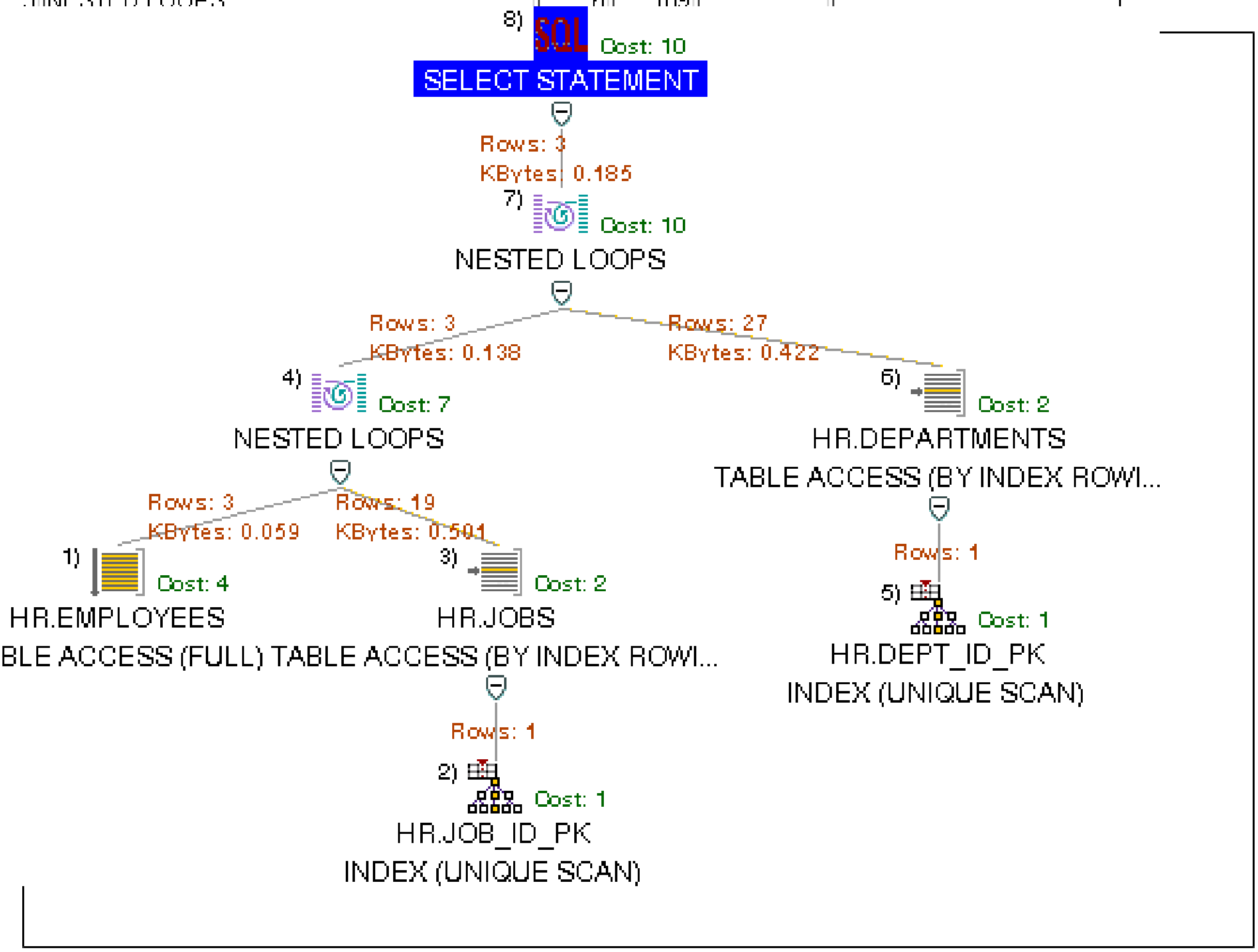
```
SQL> EXPLAIN PLAN FOR
          SELECT e.employee_id, j.job_title, e.salary, d.department_name
                FROM employees e,jobs j,departments d
                  WHERE e.employee_id < 103
                      AND e.job_id = j.job_id
                      AND e.department_id = d.department_id;

SQL> Explained
```

uery Execution Plan:

```
SQL> SELECT cardinality "ROWS", lpad(' ',level-1) ||operation        ||' ' || options||' '||
         object_name "OPERATION", cost, bytes, optimizer FROM PLAN_TABLE;
```



ROWS	OPERATION	COST	BYTES	OPTIMIZER
3	SELECT STATEMENT	6	189	CHOOSE
3	NESTED LOOPS	6	189	



dit Trail:

The data dictionary of every database has a table named SYS.AUD$, commonly referred to as the database **audit trail**, that is designed to store entries auditing database statements, privileges, or schema objects.

Setting Auditing Options

We specify auditing options using the `AUDIT` statement. The `AUDIT` statement allows you to set audit options at three levels:

Level	Effect
Statement	Causes auditing of specific SQL statements or groups of statements that affect a particular type of database object. For example, `AUDIT TABLE` audits the `CREATE TABLE`, `TRUNCATE TABLE`, `COMMENT ON TABLE`, and `DELETE [FROM] TABLE` statements.
Privilege	Audits SQL statements that are authorized by the specified system privilege. For Example, `AUDIT CREATE ANY TRIGGER` audits statements issued using the `CREATE ANY TRIGGER` system privilege.
Object	Audits specific statements on specific objects, such as `ALTER TABLE` on the `emp` table

To use the `AUDIT` statement to set statement and privilege options, user must have the `AUDIT SYSTEM` privilege. To use it to set object audit options, user must own the object to be audited or have the `AUDIT ANY` privilege.

User: sys

```
SQL> AUDIT SESSION BY hr;

SQL> AUDIT SELECT TABLE

              BY ACCESS
                      WHENEVER NOT SUCCESSFUL;
```

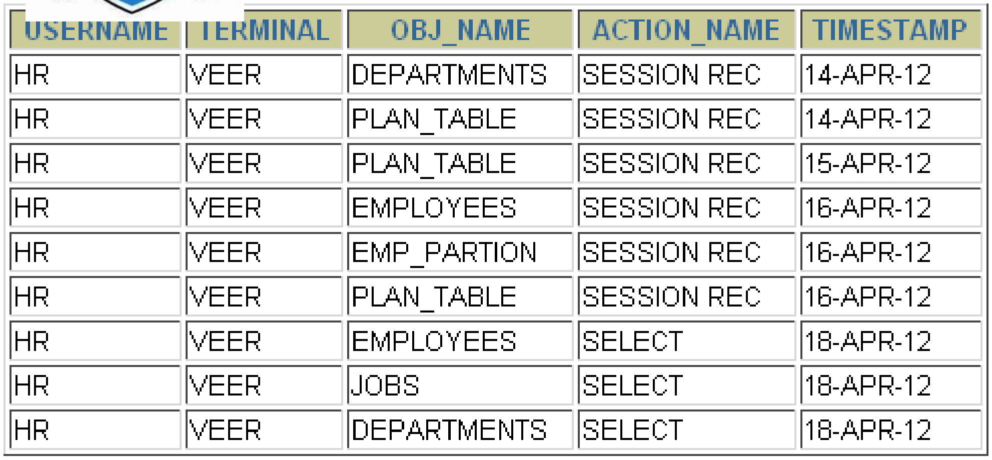
User: hr

```
SQL> SELECT e.employee_id, j.job_title, e.salary, d.department_name
          FROM employees e, jobs j, departments d
                   WHERE e.employee_id < 103
                   AND e.job_id = j.job_id
                   AND e.department_id = d.department_id;
```

dited data:

User: sys

```
SQL> SELECT username,terminal,obj_name,action_name from
           USER_AUDIT_OBJECT where username='HR';
```



USERNAME	TERMINAL	OBJ_NAME	ACTION_NAME	TIMESTAMP
HR	VEER	DEPARTMENTS	SESSION REC	14-APR-12
HR	VEER	PLAN_TABLE	SESSION REC	14-APR-12
HR	VEER	PLAN_TABLE	SESSION REC	15-APR-12
HR	VEER	EMPLOYEES	SESSION REC	16-APR-12
HR	VEER	EMP_PARTION	SESSION REC	16-APR-12
HR	VEER	PLAN_TABLE	SESSION REC	16-APR-12
HR	VEER	EMPLOYEES	SELECT	18-APR-12
HR	VEER	JOBS	SELECT	18-APR-12
HR	VEER	DEPARTMENTS	SELECT	18-APR-12

Date:-		Sign:-		Grade:-	