RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

BONAFIDE CERTIFICATE

NAME G.Meenakshi

ACADEMIC YEAR 2024-2025 SEMESTER III BRANCH AI ML-B

UNIVERSITY REGISTER No. 211623150109 7

Certified that this is the bonafide record of work done by the above student in the

database management system Laboratory during the year 2024 - 2025

Signature of Faculty - in - Charge

Submitted for the Practical Examination held on 28/11/24

External Examiner

Internal Examiner

INDEX PAGE

SL.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	MARKS	FACULTY'S SIGNATURE
01	26/7/24	CREATION OF BASE TABLE AND DML OPERATIONS	1	10	
02	30/7/24	DATA MANIPULATIONS	5	10	
03	2/8/24	WRITING BASIC SQL SELECT STATEMENT	10	10	
04	6/8/24	WORKING WITH CONSTRAINTS	14	10	
05	13/8/24	CREATING VIEWS	21	10	
06	20/8/24	RESTRICTING AND SORTING DATA	26	10	
07	27/8/24	USING SET OPERATORS	34	10	
08	3/9/24	WORKING WITH MULTIPLE TABLES	38	10	
09	10/9/24	SUB QUERIES	48	10	
10	20/9/24	AGGREGATING DATA USING GROUP FUNCTIONS	55	10	
11	24/9/24	PL SQL PROGRAMS	63	10	
12	27/9/24	WORKING WITH CURSOR PROCEDURES AND FUNCTIONS	79	10	
13	1/10/24	WORKING WITH TRIGGER	88	7	
14	8/10/24	MONGO DB	96	7	
15	18/10/24	OTHER DATABASE OBJECTS	107	7	
16	25/10/24	CONTROLLING USER ACCESS	113	7	

Ex.No.: 1	CREATION OF BASE TABLE AND DML OPERATIONS
Date: 26/7/24	

AIM:

ALGORITHM:

STEP-1: Start.

STEP-2: Create a base Table

Syntax:

CREATE TABLE <table name> (column1 type, column2 type, ...);

STEP-3: Describe the Table structure

Syntax:

DESC <table name>

STEP-4: Add a new row to a Table using INSERT statement.

Syntax:

- INSERT INTO <table name> VALUES (value1, value2..);
- INSERT INTO <table name> (column1, column2..)
  VALUES (value1, value2..);
- INSERT INTO <table name>VALUES (&column1,'&column');

STEP-5: Modify the existing rows in the base Table with UPDATE statement.

Syntax:

UPDATE <table name> SET column1=value, column2 = 'value'
WHERE (condition);

STEP-6: Remove the existing rows from the Table using DELETE statement.

Syntax:

DELETE FROM <table name> WHERE <condition>;

STEP-7: Perform a Query using SELECT statement.

Syntax:

SELECT [DISTINCT] {*,<column1,,..>} FROM <table name>
WHERE <condition>;

NAME	NULL?	TYPE
ID	Not null	Number(4)
Last_name		Varchar(25)
First_name		Varchar(25)
Userid		Varchar(25)
Salary		Number(9,2)

create table my_employee (id Number(4) constraint my_employee_id_nn not null, last_name varchar(25), first_name varchar(25), userid varchar(8), salary num(9,2));

2. Add the first and second rows data to MY_EMPLOYEE table from the following sample data.

ID	Last_name	First_name	Userid	salary
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	Cnewman	750
5	Ropebur	Audrey	aropebur	1550

insert into my_employee values(1, 'Patel', 'Ralph', 'rpatel', 895);

insert into my_employee values(2, 'Dancs', 'Betty', 'bdancs', 860);

3. Display the table with values.

Select * from my_employee;

insert

4. Populate the next two rows of data from the sample data. Concatenate the first letter of the first_name with the first seven characters of the last_name to produce Userid.

insert into my_employee values(3, 'Biri', 'Ben', 'bbiri', 1100);

insert into my_employee values(4, 'Newman', 'Chad', 'Cnewman', 750);

insert into my_employee values(5, 'Ropebur', 'Audrey', 'aropebur', 1550);

5. Delete Betty dancs from MY_EMPLOYEE table.

delete from my_employee where last_name='Dancs';

6. Empty the fourth row of the emp table.

```
Delete from my_employee where id = 4;
```

7. Make the data additions permanent.

```
commit;
```

8. Change the last name of employee 3 to Drexler.

```
update my_employee set last_name = 'Drexler'
where id = 3;
```

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

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

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 2		DATA MANIPULATIONS
Date:	30/1/24	

Create the following tables with the given structure.

EMPLOYEES TABLE

NAME	NULL?	TYPE
Employee_id	Not null	Number(6)
First_Name		Varchar(20)
Last_Name	Not null	Varchar(25)
Email	Not null	Varchar(25)
Phone_Number		Varchar(20)
Hire_date	Not null	Date
Job_id	Not null	Varchar(10)
Salary		Number(8,2)
Commission_pct		Number(2,2)
Manager_id		Number(6)
Department_id		Number(4)

(a) Find out the employee id, names, salaries of all the employees

```
select employee_id, first_name, last_name,
salary from employees;
```

(b) List out the employees who works under manager 100

```
select employee_id, first_name, last_name from
employees where manager_id = 100;
```

(c) Find the names of the employees who have a salary greater than or equal to 4800

```
select first_name, last_name from employees
where salary >= 4800;
```

(d) List out the employees whose last name is _AUSTIN'

```
select first_name, last_name from employees
where last_name ='Austin';
```

(e) Find the names of the employees who works in departments 60,70 and 80

```
select first_name, last_name from employees
where department_id in (60,70,80);
```

(f) Display the unique Manager_Id.

```
select distinct manager_id from employees;
```

Create an Emp table with the following fields: (EmpNo, EmpName, Job,Basic, DA, HRA,PF, GrossPay, NetPay) (Calculate DA as 30% of Basic and HRA as 40% of Basic)

```
create table emp(empno number(6) primary key,
empname varchar(50); job varchar(30, basic
number(30); da number(10, 2), Hra number(10,2),
pf number(10, 2), grosspay number(10,2) netpay
number(10,2));
```

(a) Insert Five **Records** and calculate GrossPay and NetPay.

(b) Display the employees whose Basic is lowest in each department.

```
select empname, job, basic from emp E
where Basic = (select min(basic) from emp
where e.job = job);
```

(c) If Net Pay is less than

a)

insert into emp(empno, empname, job, basic, da, hra, pf, grosspay, netpay) values

(1001, 'John Doe', 'Manager', 3000, 3000*0.3, 3000*0.4, 200, (3000+3000*0.3+3000*0.4), (3000+3000*0.3+3000*0.4)-200),

(1002, 'Jane Smith', 'Clerk', 2500, 2500*0.3, 2500*0.4, 150, (2500+2500*0.3+2500*0.4), (2500+2500*0.3+2500*0.4)-150),

(1003, 'Bob Johnson', 'Analyst', 4000, 4000*0.3, 4000*0.4, 250, (4000+1000*0.3+4000*0.4), (1000+4000*0.3+4000*0.4)-250);

create table dept (dept_id number(7) not null, dept_name varchar(25) not null, manager_id number(6), location_id number(4), primary key (dept_id));

2. Create the EMP table based on the following instance chart. Confirm that the table is created.

Column name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK table				
FK column				
Data Type	Number	Varchar2	Varchar2	Number
Length	7	25	25	7

create table emp (id number(7) primary key, last_name varchar(25) not null, first_name varchar(25), dept_id number(7), constraint fk_dept foreign key (dept_id) references dept(dept_id));

3 Modify the EMP table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

alter table emp modify last_name varchar(50);

4 Create the EMPLOYEES2 table based on the structure of EMPLOYEES table. Include Only the Employee_id, First_name, Last_name, Salary and Dept_id coloumns. Name the columns Id, First_name, Last_name, salary and Dept_id respectively.

create table employee as select employee_id as id, first_name, last_name, salary, dept_id, from employees

5 Drop the EMP table.

drop table emp;

6 Rename the EMPLOYEES2 table as EMP.

alter table employee rename to emp;

7 Add a comment on DEPT and EMP tables. Confirm the modification by describing the table.

comment on table dept is 'This table stores department details';
comment on table emp is 'This table stores employee details';

8 Drop the First_name column from the EMP table and confirm it.

alter table emp drop column first_name;

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 3		WRITING BASIC SQL SELECT STATEMENTS
Date:	2/8/24	

OBJECTIVES

After the completion of this exercise, the students will be able to do the following:

- List the capabilities of SQL SELECT Statement
- Execute a basic SELECT statement

Capabilities of SQL SELECT statement

A SELECT statement retrieves information from the database. Using a select statement, we can perform

- ✔ Projection: To choose the columns in a table
- ✔ Selection: To choose the rows in a table
- ✔ Joining: To bring together the data that is stored in different tables

Basic SELECT Statement

Syntax

SELECT *|DISTINCT Column_ name| alias
` FROM table_name;

NOTE:

DISTINCT—Suppress the duplicates.

Alias—gives selected columns different

headings. **Example: 1**

SELECT * FROM departments;

Example: 2

SELECT location_id, department_id FROM departments;

Writing SQL Statements

- SQL statements are not case sensitive
- SQL statements can be on one or more lines.

Using Literal Character String

- A literal is a character, a number, or a date included in the SELECT list.
- Date and character literal values must be enclosed within single quotation marks.

Example:

SELECT last_name||'is a'||job_id AS —EMPLOYEES JOBI FROM employees;

Eliminating Duplicate Rows

- Using DISTINCT keyword.

Example:

SELECT DISTINCT deparment_id FROM employees;

Displaying Table Structure

- Using DESC keyword.

Syntax

DESC table_name;

Example:

DESC employees;

Find the Solution for the following:

True OR False

1. The following statement executes successfully.

Identify the Errors

SELECT employee_id, last_name
sal*12 ANNUAL SALARY
FROM employees;

Queries

2. Show the structure of departments the table. Select all the data from it.

desc departments;

3. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

```
select employee_id, last_name, job_id,
hire_date from employees;
```

4. Provide an alias STARTDATE for the hire date.

```
Select employee_id, last_name, job_id,
hire_date as startdate from employees;
```

5. Create a query to display unique job codes from the employee table.

```
Select distinct job_id from employees;
```

6. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

```
Select last_name||', '||job_id as "employee and
title" from employees;
```

7. Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE_OUTPUT.

```
Select employee_id||', '||first_name||', '||last_name
||', '||job_id||', '||salary||', '||hire_date as
the_output from employees;
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 4	WORKING WITH CONSTRAINTS
Date: 6/8/24	

OBJECTIVE

After the completion of this exercise the students should be able to do the following

- Describe the constraints
- Create and maintain the constraints

What are Integrity constraints?

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies

The following types of integrity constraints are valid

a) **Domain Integrity**

✔ NOT NULL
✔ CHECK

b) **Entity Integrity**

✔ UNIQUE
✔ PRIMARY KEY

c) **Referential Integrity**

✔ FOREIGN KEY

Constraints can be created in either of two ways

1. At the same time as the table is created
2. After the table has been created.

Defining Constraints

Create table tablename (column_name1 data_ type constraints, column_name2 data_ type constraints ...);

Example:

Create table employlees (employee_id number(6), first_name varchar2(20), ..job_id varchar2 (10), CONSTRAINT emp_emp_id_pk PRIMARY KEY (employlee_id));

(OR)
ALTER TABLE test 1 DROP(pk, fk, col1) CASCADE CONSTRAINTS;

VIEWING CONSTRAINTS

Query the USER_CONSTRAINTS table to view all the constraints definition and names.

Example:

SELECT constraint_name, constraint_type, search_condition FROM user_constraints
WHERE table_name='employees';

Viewing the columns associated with constraints

SELECT constraint_name, constraint_type, FROM user_cons_columns
WHERE table_name='employees';

Find the Solution for the following:

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.

alter table emp add constraint my_emp_id_pk
primary key(id);

2. Create a PRIMAY KEY constraint to the DEPT table using the ID colum. The constraint should be named at creation. Name the constraint my_dept_id_pk.

alter table dept add constraint my_dept_id_pk
primary key (dept_id);

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 nonexistent deparment. Name the constraint my_emp_dept_id_fk.

alter table emp add dept_id number(4);
alter table emp add constraint my_empt_dept_id_fk
foreign key(dept_id) references dept(dept_id);

4. Modify the EMP table. Add a COMMISSION column of NUMBER data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

```
alter table emp add commission number(2,2);
alter table emp add constraint check_commission
  check (commission >0);
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 5	CREATING VIEWS
Date: 13/8/24	

After the completion of this exercise, students will be able to do the following:

- Describe a view
- Create, alter the definition of, and drop a view
- Retrieve data through a view
- Insert, update, and delete data through a view
- Create and use an inline view

View

A view is a logical table based on a table or another view. A view contains no data but is like a window through which data from tables can be viewed or changed.The tables on which a view is based are called base tables.

Advantages of Views

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

Classification of views

1. Simple view
2. Complex view

Feature	Simple	Complex
No. of tables	One	One or more
Contains functions	No	Yes
Contains groups of data	No	Yes
DML operations thr' view	Yes	Not always

Creating a view

Syntax

Use of WITH READ ONLY option.
Any attempt to perform a DML on any row in the view results in an oracle server error.

Try this code:

```
CREATE OR REPLACE VIEW empvu10(employee_number, employee_name,job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id=10
WITH READ ONLY;
```

Find the Solution for the following:

1. Create a view called EMPLOYEE_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
create view employee_vu as select employee_id
first_name || ' ' || last_name as employee, department_id
from employee;
```

2. Display the contents of the EMPLOYEES_VU view.

```
select * from employee_vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary views.

```
select view_name, text from user_views
where view_name = 'employee_vu';
```

4. Using your EMPLOYEES_VU view, enter a query to display all employees names and department.

```
select employee, department_id from
employee_vu;
```

Use of WITH READ ONLY option.
Any attempt to perform a DML on any row in the view results in an oracle server error.

Try this code:

```
CREATE OR REPLACE VIEW empvu10(employee_number, employee_name,job_title)
AS SELECT employee_id, last_name, job_id
FROM employees
WHERE department_id=10
WITH READ ONLY;
```

Find the Solution for the following:

1. Create a view called EMPLOYEE_VU based on the employee numbers, employee names and department numbers from the EMPLOYEES table. Change the heading for the employee name to EMPLOYEE.

```
create view employee_vu as select employee_id
first_name || ' ' || last_name as employee, department_id
from employee;
```

2. Display the contents of the EMPLOYEES_VU view.

```
select * from employee_vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary views.

```
select view_name, text from user_views
where view_name = 'employee_vu';
```

4. Using your EMPLOYEES_VU view, enter a query to display all employees names and department.

```
select employee, department_id from
employee_vu;
```

5. Create a view named DEPT50 that contains the employee number, 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 employee_id as
empno, last_name as employee, department_id as
from employees where department_id = 50 with
                                    read only;
```

6. Display the structure and contents of the DEPT50 view.

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

7. Attempt to reassign Matos to department 80.

```
update dept50 set deptno = 80 where
employee = 'Matos';
```

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the Employees, DEPARTMENTS and JOB_GRADE tables. Label the column Employee, Department, salary, and Grade respectively.

```
create view salary_vu as select e.last_name as
employee, d.department_name as department,
e.salary as salary, j.grade_level as grade
from employee e join departments d on
e.department_id = d.department_id
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 6		RESTRICTING AND SORTING DATA
Date:	20/8/24	

After the completion of this exercise, the students will be able to do the following:

- Limit the rows retrieved by the queries
- Sort the rows retrieved by the queries
-

Limiting the Rows selected

- Using WHERE clause
- Alias cannot used in WHERE clause

Syntax

SELECT----------
FROM- - - - - - -
WHERE condition;

Example:

SELECT employee_id,last_name, job_id, deparment_id FROM employees WHERE department_id=90;

Character strings and Dates

Character strings and date values are enclosed in single quotation marks.

Character values are case sensitive and date values are format sensitive.

Example:

SELECT employee_id,last_name, job_id, deparment_id FROM employees
WHERE last_name='WHALEN';

Comparison Conditions

All relational operators can be used. (=, >, >=, <, <= ,<>,!=)

Example:

SELECT last_name, salary

```
SELECT last_name, salary*12 annsal , job_id,department_id,hire_date
FROM employees
ORDER BY annsal;
```

Example:4

Sorting by Multiple columns

```
SELECT last_name, salary , job_id,department_id,hire_date
FROM employees
ORDER BY department_id, salary DESC;
```

Find the Solution for the following:

1. Create a query to display the last name and salary of employees earning more than 12000.

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

2. Create a query to display the employee last name and department number for employee number 176.

```
select last_name, department_id from
employees where employee_id = 176;
```

3. Create a query to display the last name and salary of employees whose salary is not in the range of 5000 and 12000. (hints: not between)

```
select last_name, salary from employees
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.(hints: between)

```
select last_name, job_id, hire_date from
employees where hire_date between
'20-Feb-1998' and '01-may-1998' order by
hire_date asc;
```

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.(hints: in, orderby)

```
select last_name , department_id from
employees where department_id in
(20,50) order by last_name asc;
```

6. Display the last name and salary of all employees who earn between 5000 and 12000 and are in departments 20 and 50 in alphabetical order by name. Label the columns EMPLOYEE, MONTHLY SALARY respectively.(hints: between, in)

```
select last_name as employee , salary
as "monthly salary" from employees
where salary between 5000 and 12000
and department_id in (20,50) order by
last_name asc;
```

7. Display the last name and hire date of every employee who was hired in 1994.(hints: like)

```
select last_name , hire_date from
employees where hire_date like '%1994';
```

8. Display the last name and job title of all employees who do not have a manager.(hints: is null)

```
select last_name, job_id from employees
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.(hints: is not nul,orderby)

```
select last_name, salary, commission
from employees where commission is
not null order by salary desc,
commission desc;
```

10. Display the last name of all employees where the third letter of the name is *a*.(hints:like)

```
select last_name from employees
where last_name like '__a%';
```

11. Display the last name of all employees who have an a and an *e* in their last name.(hints: like)

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

12. Display the last name and job and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to 2500 ,3500 or 7000.(hints:in,not in)

```
select last_name, job_id, salary from employees
where job_id in ('sales_rep', 'stock_clerk')
and salary not in
(2500, 3500, 7000);
```

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

Ex.No.: 7		USING SET OPERATORS
Date:	27/8/24	

Objectives

After the completion this exercise, the students should be able to do the following:

- Describe set operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned

The set operators combine the results of two or more component queries into one result.

Queries containing set operators are called *compound queries*.

Operator	Returns
UNION	All distinct rows selected by either query
UNION ALL	All rows selected by either query, including all duplicates
INTERSECT	All distinct rows selected by both queries
MINUS	All distinct rows that are selected by the first SELECT statement and not selected in the second SELECT statement

The tables used in this lesson are:

- EMPLOYEES: Provides details regarding all current employees

- JOB_HISTORY: Records the details of the start date and end date of the former job, and the job identification number and department when an employee switches jobs

UNION Operator

Guidelines

- The number of columns and the data types of the columns being selected must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.

- UNION operates over all of the columns being selected.

- NULL values are not ignored during duplicate checking.

- The IN operator has a higher precedence than the UNION operator.

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired (that is, they changed jobs but have now gone back to
doing their original job).

```
SELECT employee_id, job_id FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

Example

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

MINUS Operator

Guidelines

- The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns need not be identical.
- All of the columns in the WHERE clause must be in the SELECT clause for the MINUS operator to work.

Example:

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id,job_id
FROM employees
MINUS
SELECT employee_id,job_id
FROM job_history;
```

Find the Solution for the following:

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use set operators to create this report.

```
select department_id from departments minus
select departmentid from employees where
  job_id = 'ST_CLERK';
```

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use set operators to create this report.

```
select country_id, country_name from countries minus
select country_id, country_name from countries c
join locations l on c.country_id = l.country_id
join departments d on l.location_id = d.location_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, department_id from employees where
department_id=10 union select job_id, department_id
from employees where department_id =50 union
select job_id, department_id from employees where
department_id=20;
```

4. Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs but have now gone back to doing their original job).

```
select employee_id, job_id from employees where
hire_date = (select min(hire_date) from employees
where employee_id = employees.employee_id
intersect select employee_id, job_id from employees;
```

5. The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.

- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound query to accomplish this.

Evaluation Procedure	Marks awarded
Query(5)	5
Execution (5)	5
Viva(5)	5
Total (15)	15
Faculty Signature	

5)
```
select e.last_name, e.department_id, d.department_name
from employees e full outer join departments d
on e.department_id = d.department_id;
```

Ex.No.: 8	WORKING WITH MULTIPLE TABLES
Date: 3/9/24	

Objective

After the completion of this exercise, the students will be able to do the following:
• Write SELECT statements to access data from more than one table using equality and nonequality joins
• View data that generally does not meet a join condition by using outer joins
• Join a table to itself by using a self join
Sometimes you need to use data from more than one table.

Cartesian Products
• A Cartesian product is formed when:
– A join condition is omitted
– A join condition is invalid
– All rows in the first table are joined to all rows in the second table
• To avoid a Cartesian product, always include a valid join condition in a WHERE clause.
A Cartesian product tends to generate a large number of rows, and the result is rarely useful. You should always include a valid join condition in a WHERE clause, unless you have a specific need to combine all rows from all tables.
Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.
Example:
To displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables.

```
SELECT last_name, department_name dept_name
FROM  employees, departments;
```

Types of Joins

• Equijoin
• Non-equijoin
• Outer join
• Self join
• Cross joins
• Natural joins
• Using clause
• Full or two sided outer joins
• Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

```
SELECT table1.column, table2.column
```

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM  employees e, departments d
WHERE  d.department_id = e.department_id  (+);
```

FULL OUTER JOIN
Example:

```
SELECT e.last_name, e.department_id, d.department_name
FROM  employees e
FULL OUTER JOIN departments d
ON  (e.department_id = d.department_id) ;
```

This query retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table. It alslso retrieves all rows in the DEPARTMENTS table, even if there is no match in the EMPLOYEES table.

Find the Solution for the following:

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

```
select e.last_name, e.dept_id, d.dept_name from
employees e join
join depts d on e.dept_id=d.dept_id;
```

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

```
select distinct e.job_id, d.dept_name, l.loc_id, l.city
from employees e
join depts d on e.dept_id=d.dept_id
join loc l on d.loc_id=l.loc_id where e.dept_id=80;
```

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, l.loc_id, l.city
from employees e
join depts d on e.dept_id = d.dept_id
join loc l on d.loc_id = l.loc_id where
e.commission_pct is not null;
```

4 Display the employee last name and department name for all employees who have an a(lowercase) in their last names. P

```
select e.last_name, d.dept_name from employees e
join depts d on e.dept_id = d.dept_id where e.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.job_id, d.dept_id, d.dept_name
from employees e
join depts d on e.dept_id = d.dept_id
join locs l on d.loc_id = l.loc_id where l.city =
'Toronto';
```

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 as "Employee", e.employee_id as
"Emp#", m.last_name as "manager", m.employee_id as
"Mgr#" from employees e left join employees m on
e.manager_id = m.employee_id;
```

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

```
select e.last_name as "Employee", e.employee_id as "Emp#",
m.last_name as "manager", m.employee_id as "Mgr#" from
employee e left join employees m on e.manager_id =
m.employee_id order by e.employee_id;
```

8. 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 e1.last_name as "Employee", e1.dept_id as "Dept#",
e2.last_name as "colleague" from employees e1
join employees e2 on e1.dept_id = e2.dept_id where
e1.employee_id = e2.employee_id;
```

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

```
desc job_grades;
select e.last_name, e.job_id, d.dept_name, e.salary,
j.grade_level from employees e
joins dept d on e.dept_id = d.dept_id;
```

10. Create a query to display the name and hire date of any employee hired after employee Davies.

```
select e.last_name, e.hire_date from employees e
where e.hire_date > (select hire_date from employee
where last_name = 'Davies');
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively.

```
Select e.lastname as "Employee", e.hire_date as "Emp
Hired", m.last_name as "Manager", m.hire_date as
"Mgr Hired" from employees e
join employees m on e.manager_id = m.employee_id
where e.hire_date < m.hire_date;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 9		SUB QUERIES
Date:	10/9/24	

Objectives

After completing this lesson, you should be able to do the following:
- Define subqueries
- Describe the types of problems that subqueries can solve
- List the types of subqueries
- Write single-row and multiple-row subqueries

Using a Subquery to Solve a Problem
Who has a salary greater than Abel's?

Main query:
Which employees have salaries greater than Abel's salary?

Subquery:
What is Abel's salary?

Subquery Syntax

SELECT *select_list* FROM *table* WHERE *expr operator* (SELECT *select_list* FROM *table*);

- The subquery (inner query) executes once before the main query (outer query).

- The result of the subquery is used by the main query.

A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including the following:

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:
operator includes a comparison condition such as >, =, or IN

Note: Comparison conditions fall into two classes: single-row operators

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

Notice that the null value as part of the results set of a subquery is not a problem if you use the IN operator. The IN operator is equivalent to =ANY. For example, to display the employees who have subordinates, use the following SQL statement:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id IN (SELECT mgr.manager_id FROM employees mgr);
```

Display all employees who do not have any subordinates:

```
SELECT last_name FROM employees
WHERE employee_id NOT IN (SELECT manager_id FROM employees WHERE manager_id
IS NOT NULL);
```

Find the Solution for the following:

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees who work with Zlotkey (excluding Zlotkey).

```
define emp_last_name = '&Employee_last_name';
select e.last_name, e.hire_date from employees e
join employees ref_emp on e.dept_id=ref_emp.dept_id
where ref_emp.last_name = '&emp_last_name'
```

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

```
select employee_id, last_name, salary from employees
where salary > (select avg(salary) from employees)
order by salary asc;
```

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*.

```
select employee_id, e.last_name from employees e
where e.dept_id in (select dept_id from
employees where last_name like '%u%');
```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
select e.last_name, e.dept_id, e.job_id from
employees e join depts d on e.dept_id=d.dept_id
where d.loc_id = 1700;
```

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
select e.last_name, e.salary from employees e
join employees m on e.manager_id = m.employee_id
where m.last_name = 'King';
```

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
select e.dept_id, e.last_name, e.job_id from
employees e join depts d on e.dept_id = d.dept_id
where d.dept_name = 'Executive';
```

7. Modify the query 3 to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a *u*.

```
select e.employee_id, e.last_name, e.salary from
employees e where e.salary > (select avg(salary)
from employees) and e.dept_id in (select
dept_id from employees where last_name like
'%u%');
```

Ex.No.: 10	AGGREGATING DATA USING GROUP FUNCTIONS
Date: 20/9/24	

Objectives

After the completion of this exercise, the students be will be able to do the following:

- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

What Are Group Functions?

Group functions operate on sets of rows to give one result per group

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG([DISTINCT|ALL] *n*)	Average value of *n*, ignoring null values
COUNT({*| [DISTINCT|ALL] *expr*})	Number of rows, where *expr* evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX([DISTINCT|ALL] *expr*)	Maximum value of *expr*, ignoring null values
MIN([DISTINCT|ALL] *expr*)	Minimum value of *expr*, ignoring null values
STDDEV([DISTINCT|ALL] *x*)	Standard deviation of *n*, ignoring null values
SUM([DISTINCT|ALL] *n*)	Sum values of *n*, ignoring null values
VARIANCE([DISTINCT|ALL] *x*)	Variance of *n*, ignoring null values

Group Functions: Syntax

SELECT [*column*,] *group_function(column)*, ...
FROM *table*
[WHERE *condition*]

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

SELECT MAX(AVG(salary)) FROM employees GROUP BY department_id;

Summary

In this exercise, students should have learned how to:

- Use the group functions COUNT, MAX, MIN, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

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

Find the Solution for the following:

Determine the validity of the following three statements. Circle either True or False.

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

2. Group functions include nulls in calculations.
True/False False

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

The HR department needs the following reports:

4. Find 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 round(max(salary)) as maxi, round(min(salary)
as mini, round(sum(salary)) as sum, round(avg(salary)
as avge from employees ;
```

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

```
select job_id, round(max(salary)) as maxi,
round(min(salary)) as mini, round(sum(salary))
as sum, round(avg(salary)) as avge from
employee group by job_id;
```

6. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

```
define job_title = '&Enter_job_title';
select job_id, count(*) as no_of_people from employees
where job_id = '&job_title' group by job_id;
```

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(distinct manager_id) as no_of_managers
from employees where manager_id is not null;
```

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
select (max(salary) - min(salary)) as difference
from employees;
```

9. Create a report to 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 manager_id, min(salary) as salary from
employees where manager_id is not null group by
manager_id having min(salary) > 6000 order by
salary desc;
```

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
select count(*) as tot_employees,
sum(case when to_char(hire_date, 'YYYY')='1995'
        then 1 else 0 end)as hired_in_1995,
sum(case when to_char(hire_date,'YYYY')='1996'
        then 1 else 0 end) as hired_in_1996,
```

4)

```
select job_id,
sum(case when dept_id = 20 then salary else 0
    end) as dept_20_salary,
sum(case when dept_id = 50 then salary else 0
    end) as dept_50_salary,
sum(ccase when dept_id = 80 then salary else 0
    end) as dept_80_salary,
sum(ccase when dept_id = 90 then salary else 0
    end) as dept_90_salary,
from employees where dept_id in (20,50,80,90)
group by job_id;
```

```
sum(case when to_char(hire_date,'YYYY')='1997' then
        1 else 0 end) as hired_in_1997,
sum(case when to_char(hire_date,'YYYY')='1998'
        then 1 else 0 end) as hired_in_1998
from employees;
```

11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

12. Write a query to display each department's name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

```
select d.dept_name as dept_name, l.city as loc,
count(e.employee_id) as no_of_people, round(avg(
e.salary),2) as avg_salary from depts d
left join employees e on d.dept_id = e.dept_id
left join locs l on d.location_id = l.location_id
group by d.dept_name, l.city;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 11		PL SQL PROGRAMS
Date:	24/9/24	

PROGRAMS

TO DISPLAY HELLO MESSAGE

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:='Hello';
  5  dbms_output.put_line(a);
  6  end;
  7  /
Hello

PL/SQL procedure successfully completed.
```

TO INPUT A VALUE FROM THE USER AND DISPLAY IT

```
SQL> set serveroutput on;
SQL> declare
  2  a varchar2(20);
  3  begin
  4  a:=&a;
  5  dbms_output.put_line(a);
  6  end;
  7  /
Enter value for a: 5
old  4: a:=&a;
new  4: a:=5;
5

PL/SQL procedure successfully completed.
```

GREATEST OF TWO NUMBERS

```
SQL> set serveroutput on;

SQL> declare
  2  a number(7);
```

PROGRAM 1

Write a PL/SQL block to calculate the incentive of an employee whose ID is 110.

```
DECLARE
        v_salary employees.salary%TYPE;
        v_incentive NUMBER(8,2);
BEGIN
        select salary into v_salary from employees
        where employee_id =110;
        v_incentive = v_salary*0.10;
        DBMS_output.put_line('Incentive for
                Employee ID 110: '||v_incentive);
END;
```

PROGRAM 2

Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

```
DECLARE
        "Myvar" NUMBER :=10;
BEGIN
        --Invalid Reference without Quotes
        DBMS_output.put_line(Myvar);
        DBMS_output.put_line("Myvar");
END;
```

PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID 122.
Sample table: employees

```
DECLARE
        v_current_salary employees.salary%TYPE;
        v_new_salary employees.salary%.TYPE;
BEGIN
        select salary into v_current_salary
            from employees where employee_id = 122;
        v_new_salary = v_current_salary*1.05;
        update employees
        set salary = v_new_salary
        where employee_id = 122;
        DBMS_output
```

PROGRAM 4

Write a PL/SQL block to create a procedure using the "IS [NOT] NULL Operator" and show AND operator returns TRUE if and only if both operands are TRUE.

```
create or replace procedure check_employee_info
    (emp_id in number) is
    v_salary employees.salary%TYPE;
    v_commission employees.commission_pct%TYPE;
BEGIN
    select salary, commission_pct into v_salary
    v_commission from employees where employee_id =
                                        emp_id;
    if v_salary is not null & v_commission is not null then
        dbms_output.put_line('Both salary and commission
                              are avail for emp ID')
    else
        dbms_output.put_line('One or both values are
                              missing for employee ID:' ||emp_id);
    end if;
END;
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
DECLARE
        v_employee_name employees.last_name%TYPE;
BEGIN
        FOR employee IN(select last_name from
        employees where last_name like '%a%'
        escape '\') loop
        v_employee_name = employee.last_name;
        dbms_output.put_line('Emp name with
                                letter "a": '||v_emp_name);
    END LOOP;
END;
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.

```
DECLARE
        num1 NUMBER = 10;
        num2 NUMBER = 20;
        num_small NUMBER;
        num_large NUMBER;
BEGIN
        if num1 < num2 then
                num_small = num1;
                num_large = num2;
        else
                num_small = num2;
                num_large = num1;
        end if;
        dbms_output.put_line('small no '||num_small);
        dbms_output.put_line('large no '|| num_large);
END;
```

PROGRAM 5

Write a PL/SQL block to describe the usage of LIKE operator including wildcard characters and escape character.

```
DECLARE
        v_employee_name employees.last_name%TYPE;
BEGIN
        FOR employee IN(select last_name from
        employees where last_name like '%a%'
        escape '\') loop
        v_employee_name = employee.last_name;
        dbms_output.put_line('Emp name with
                                letter 'a': '||v_emp_name);
    END LOOP;
END;
```

PROGRAM 6

Write a PL/SQL program to arrange the number of two variable in such a way that the small number will store in num_small variable and large number will store in num_large variable.

```
DECLARE
        num1 NUMBER = 10;
        num2 NUMBER = 20;
        num_small NUMBER;
        num_large NUMBER;

BEGIN
        if num1 < num2 then
                num_small = num1;
                num_large = num2;

        else
                num_small = num2;
                num_large = num1;
        end if;
        dbms_output.put_line('small no '||num_small);
        dbms_output.put_line('large no '||num_large);

END;
```

PROGRAM 9

Write a PL/SQL program to count number of employees in department 50 and check whether this department have any vacancies or not. There are 45 vacancies in this department.

```
DECLARE
  V_emp_count NUMBER; -Variable to store the count of employees
  in department 50
  V_vacancies NUMBER:=45; -Total vacancies in department 50
BEGIN
  - Count the number of employees in department 50
    SELECT COUNT(*) INTO V_emp_count
    FROM employees
    WHERE department_id = 50;
 - Check if the department has vacancies
 IF V_emp_count < V_vacancies THEN
    DBMS_OUTPUT.PUT_LINE ('Department has 50 vacancies.
       Available vacancies: ' || (V_vacancies - V_emp_count));
ELSE DBMS_OUTPUT.PUT_LINE ('Deparment 50 has no vacancies.');
  END IF;
  END;
```

PROGRAM 10

Write a PL/SQL program to count number of employees in a specific department and check whether this department have any vacancies or not. If any vacancies, how many vacancies are in that department.

```
DECLARE
  P_dept_id NUMBER:=50; -Department ID(can be changed for other departments)
  V_emp_count NUMBER; -Variable to store the count of employees in the
  V_vacancies NUMBER:=45; - Total Vacancies in the department  department
BEGIN
  - Count the number of employees in the given department
  SELECT COUNT(*) INTO V_emp_count
  FROM employees
  WHERE department_id=p_dept_id;
- Check if the department has Vacancies
  IF V_emp_count<V_vacancies THEN
     DBMS_OUTPUT.PUT_LINE('Department' || p_dept_id || ' has Vacancies.
        Available Vacancies: ' || (V_vacancies-V_emp_count));
  ELSE
     DBMS_OUTPUT.PUT_LINE('Department' || p_dept_id || ' has NO
                                                    Vacancies.');
    END IF;
    END;
    /
```

PROGRAM 11

Write a PL/SQL program to display the employee IDs, names, job titles, hire dates, and salaries of all employees.

```
BEGIN
  FOR emp IN (SELECT employee_id,first_name,last_name,job_id,
    hire_date,salary FROM employees) LOOP
 DBMS_OUTPUT.PUT_LINE('Employee ID:' || emp.employee_id ||
   ', Name:' || emp.first_name || ' ' || emp.last_name ||
   ', Job Title:' || emp.job_id ||
   ', Hire Date:' || TO_CHAR(emp.hire_date,'YYYY-MM-DD') ||
   ', Salary:' || emp.salary);
 END LOOP;
  END;
   /
```

PROGRAM 12

Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

```
BEGIN
  FOR emp IN (SELECT e.employee_id, e.first_name, e.last_name,
    d.department_name
  FROM employees e
  JOIN departments d ON e.department_id = d.department_id)
                                              LOOP
DBMS_OUTPUT.PUT_LINE('Employee ID:' || emp.employee_id ||
   ', Name:' || emp.first_name || ' ' || emp.last_name ||
   ', Department:' || emp.department_name);
  END LOOP;
  END;
  /
```

PROGRAM 13
Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

```
BEGIN
  FOR job IN (SELECT job_id, job_title, min_salary
    FROM jobs) LOOP
DBMS_OUTPUT.PUT_LINE('Job ID: ' || job.job_id ||
  ', Job Title: ' || job.job_title ||
  ', Minimum Salary: ' || job.min_salary);
END LOOP;
  END;
  /
```

PROGRAM 14
Write a PL/SQL program to display the employee IDs, names, and job history start dates of all employees.

```
BEGIN
  FOR emp IN (SELECT e.employee_id, e.first_name, e.last_name,
    FROM employees e                          j.start_date
  JOIN job_history j ON e.employee_id=j.employee_id) LOOP
DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp.employee_id ||
  ', Name: ' || emp.first_name || ' ' || emp.last_name ||
  ', Job History Start Date: ' || TO_CHAR(emp.start_date,
                                  'YYYY-MM-DD'));
  END LOOP;
    END;
    /
```

PROGRAM 15

Write a PL/SQL program to display the employee IDs, names, and job history end dates of all employees.

```
BEGIN
   FOR emp IN (SELECT e.employee_id, e.first_name, e.last_name,
      FROM employees e                          j.end_date
   JOIN job_history j ON e.employee_id = j.employee_id) LOOP
DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp.employee_id ||
      ', Name: ' || emp.first_name || ' ' emp.last_name ||
      ', Job History End Date: ' || TO_CHAR(emp.end_date,
                                    'YYYY-MM-DD'));
   END LOOP;
    END;
   /
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

PROGRAM 4

```
CREATE OR REPLACE PROCEDURE Calculate_incentive (
    P_emp_id IN NUMBER, - Employee ID
    P_target_achieved IN NUMBER, - Target achieved by the employee
    P_target_figure IN NUMBER - Target figure (expected target)
) IS
    V_incentive NUMBER := 0; - Variable to store the calculated
                                incentive
    V_message VARCHAR2(100); - Variable to store the message
BEGIN
    - Calculate the incentive based on the target achieved
    IF P_target_achieved >= P_target_figure THEN
        - If target is achieved or exceeded, calculate 10% incentive
        V_incentive := P_target_achieved * 0.10;
    ELSE
        - If target is not achieved, no incentive
        V_incentive := 0;
    END IF;
    - Update the incentive in the employee's record
    UPDATE employees
    SET incentive = V_incentive
    WHERE emp_id = P_emp_id;
    - Check if the record was updated
    IF SQL%ROWCOUNT > 0 THEN
        V_message := 'Record updated successfully.';
    ELSE
        V_message := 'Record not updated. Employee ID not found or no
                      change in incentive.';
    END IF;
    - Display the message
    DBMS_OUTPUT.PUT_LINE(V_message);
    EXCEPTION
        WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END calculate_incentive;
/
```

PROGRAM 8

```
CREATE OR REPLACE PROCEDURE calculate_incentive_by_sale(
    P_emp_id IN NUMBER, - Employee ID
    P_sales_amount IN NUMBER - Sales amount achieved by
      the employee ) IS

    V_incentive NUMBER := 0; - Variable to store the calculated
                                incentive
    V_message VARCHAR2 (100); - Variable to store the message
BEGIN
  - Calculate incentive based on the sales amount
  IF P_sales_amount >= 100000 THEN
    - For sales >= 100000, 15% incentive
    V_incentive := P_sales_amount * 0.15;
  ELSIF P_sales_amount >= 50000 AND P_sales_amount
                                  < 100000 THEN
    - For sales between 50000 and 99999, 10% incentive
    V_incentive := P_sales_amount * 0.10;
  ELSIF P_sales_amount >= 20000 AND P_sales_amount <
                                  50000 THEN
    - For sales between 20000 and 49999, 5% incentive
    V_incentive := P_sales_amount * 0.05;
  ELSE
    - FOR sales less than 20000, no incentive
    V_incentive := 0 ;
  END IF;
```

- Update the employee's incentive in the employee table

```
UPDATE employees
SET incentive = v_incentive
WHERE emp_id = p_emp_id;
```

- Check if the record was updated

```
IF SQL%ROWCOUNT > 0 THEN
   v_message := 'Incentive calculated and record updated Successfully.';
ELSE
   v_message := 'Employee not found or no change in incentive.';
END IF;
```

- Display the result message

```
DBMS_OUTPUT.PUT_LINE(v_message);

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);

END calculate_incentive_by_sale;
/
```

Ex.No.: 12		WORKING WITH CURSOR, PROCEDURES AND FUNCTIONS
Date:	27/9/24	

AIM:

Create PL/SQL Blocks to perform the Item Transaction Operations using CURSOR, FUNCTION and PROCEDUERE.

ALGORITHM:

STEP-1: Start.

STEP-2: Create two tables Item Master and Item Trans.

itemmaster(itemid , itemname, stockonhand)

itemtrans(itemid ,itemname ,dateofpurchase ,quantity)

STEP-3: Create a PROCEDURE with id, name and quantity as parameters which make a call to the FUNCTION by passing id, name, dop, and quantity as parameters dop is set as sysdate.

STEP-4: Using FUNCTION fetch each record from the table Item Master using CURSOR inside a Loop statement,

If Item Master's ItemId is equal to the entered ID value then exit the loop otherwise fetch the next record.

```
loop
        fetch master into masterrec
        exit when master%notfound
        if masterrec.itemid=id then
   exit;
  end if;
 end loop;
```

STEP-5: If Itemmaster's itemid = id then,

Add the Itemmaster's stockonhand with the given quantity and update the ItemMaster table and insert the Item information into the ItemTrans table.

STEP-6: Else, if the inputed item is not present in the ItemMaster table then insert the

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

```
- Create the function to calculate factorial
CREATE OR REPLACE FUNCTION factorial (n IN NUMBER)
   RETURN NUMBER IS result NUMBER := 1;
BEGIN
   - Check if the input is a positive number or zero
   IF n<0 THEN
      RETURN -1; - Return -1 for invalid input
   ELSIF n=0 THEN                    (negative number)
      RETURN 1; - Factorial of 0 is 1
   ELSE
      - LOOP to calculate factorial
     FOR i IN 1..n LOOP
       result := result * i;
     END LOOP;
     RETURN result;
  END IF;
  END;
  /
    - Test the function by calling it
    DECLARE
      num NUMBER := 5; - Change this value to test
      with different numbers fact NUMBER;
  BEGIN
     - Call the factorial function
    fact := factorial(num);
    - Output the result
     DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || '
   END;                                  is: ' || fact);
   /
```

PROGRAM 2

```
CREATE OR REPLACE PROCEDURE get_book_info (
    P_book_id IN NUMBER,
    P_book_title INOUT VARCHAR2,
    P_author OUT VARCHAR 2,
    P_Published_year OUT NUMBER,
    P_genre OUT VARCHAR 2 ) IS

BEGIN
        SELECT book_title, author, published_year, genre
        INTO P_book_title, P_author, P_published_year, P_genre
        FROM books
        WHERE book_id = P_book_id;

        IF P_book_title IS NULL OR P_book_title = '' THEN
            P_book_title := 'Unknown Title'; -

        END IF;

    EXCEPTION
        WHEN NO_DATA_FOUND THEN
        P_author := 'Not Found';
        P_Published_year := NULL;
        P_genre := 'Not Found';

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' ||
                                        SQLERRM);

    END;
    /
```

```
DECLARE
    V_book_title VARCHAR2(255);
    V_author VARCHAR2(255);
    V_Published_Year NUMBER;
    V_genre VARCHAR2(255);
    V_book_id NUMBER := 101;

BEGIN
    V_book_title := '';
    get_book_info(V_book_id, V_book_title, V_author,
                  V_published_year, V_genre);

    DBMS_OUTPUT.PUT_LINE('Book Title: ' || V_book_title);
    DBMS_OUTPUT.PUT_LINE('Author: ' || V_author);
    DBMS_OUTPUT.PUT_LINE('Published Year: ' || V_Published_Year);
    DBMS_OUTPUT.PUT_LINE('Genre: ' || V_genre);

    END;
```

Ex.No.: 13		WORKING WITH TRIGGER **TRIGGER**
Date:	1/10/24	

DEFINITION

A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database. The parts of a trigger are,

- **Trigger statement**: Specifies the DML statements and fires the trigger body. It also specifies the table to which the trigger is associated.
- **Trigger body or trigger action**: It is a PL/SQL block that is executed when the triggering statement is used.
- **Trigger restriction**: Restrictions on the trigger can be achieved

The different uses of triggers are as follows,

- *To generate data automatically*
- *To enforce complex integrity constraints*
- *To customize complex securing authorizations*
- *To maintain the replicate table*
- To audit data modifications

TYPES OF TRIGGERS

The various types of triggers are as follows,

- **Before**: It fires the trigger before executing the trigger statement.
- **After**: It fires the trigger after executing the trigger statement
- .
- **For each row**: It specifies that the trigger fires once per row
- .
- **For each statement**: This is the default trigger that is invoked. It specifies that the trigger fires once per statement.

VARIABLES USED IN TRIGGERS

- :new
- :old

Program 1

Write a code in PL/SQL to develop a trigger that enforces referential integrity by preventing the deletion of a parent record if child records exist.

```
create or replace trigger prevent_parent_deletion
before delete on parent_table
for each row
declare
    v_count number;
begin
    select count(*) into v_count
    from child_table
    where parent_id = old.parent_id;
    if v_count > 0 then
        raise_application_error
    end if;
end;
```

Program 2

Write a code in PL/SQL to create a trigger that checks for duplicate values in a specific column and raises an exception if found.

```
create or replace trigger check_dup_values
before insert or update on eg_table
for each row
declare
    v_count number;
begin
    select count(*) int v_value
    from example_table
    where column_name = new.column_name;
    if v_count > 0 then
        raise_application_error
    end if;
end;
```

Program 3

Write a code in PL/SQL to create a trigger that restricts the insertion of new rows if the total of a column's values exceeds a certain threshold.

```
create or replace trigger restrict_ins
before insert on sales_table
for each row
declare
        v_total_sales NUMBER;
        v_threshold NUMBER = 100000;
begin
        select sum(sales_amt) into
        from sales_table;                v_total_sales
    if (v_total_sales + new.sales_amt) >
                                    v_threshold
      then raise_app_error
    end if;
end;
```

Program 4

Write a code in PL/SQL to design a trigger that captures changes made to specific columns and logs them in an audit table.

```
create or replace trigger audit_emp
after update on emp_table
for each row
declare
        v_audit_id NUMBER;
begin
    If old.emp_name != new.emp_name then
      insert into audit_table (audit_id,
                    table_name, column_name)
      values (audit.seq.nextval, 'emp_table',
                                 'emp_name');
    end if;
    if old.salary != new.salary then
      insert into audit_table (audit_id, table_name,
                                    column_name)
      values (audit_seq.nextval, 'emp_table',
                                 'salary');
      end if;
end;
```

Program 5

Write a code in PL/SQL to implement a trigger that records user activity (inserts, updates, deletes) in an audit log for a given set of tables.

```
create or replace trigger audit_emp
after insert or update or delete on emp_table
for each row
declare
        v_change_type varchar2(10);
begin
    if inserting then
        v_change_type = 'insert';
        insert into user_act_audit(audit_id,
                    table_name, op_type, column_type)
        values (audit_seq.nextval, 'emp_table', v_change_type,
                    null);
    else if deleting then
        insert into user_act_audit(audit_id, table_name,
                    op_type, column_name)
    values (audit_seq.nextval, 'emp_table', v_change_type);
    endif;
end;
```

Program 7

Write a code in PL/SQL to implement a trigger that automatically calculates and updates a running total column for a table whenever new rows are inserted.

```
create or replace trigger update_run
after insert on sales_table
for each row
declare
        v_tot number;
begin
        select nvl(max(run_total), 0) + new.sale_amt
        int v_tot from sales_table;
        update sales_table
        set running_tot = v_tot
        where sale_id = new.sale_id;

end;
```

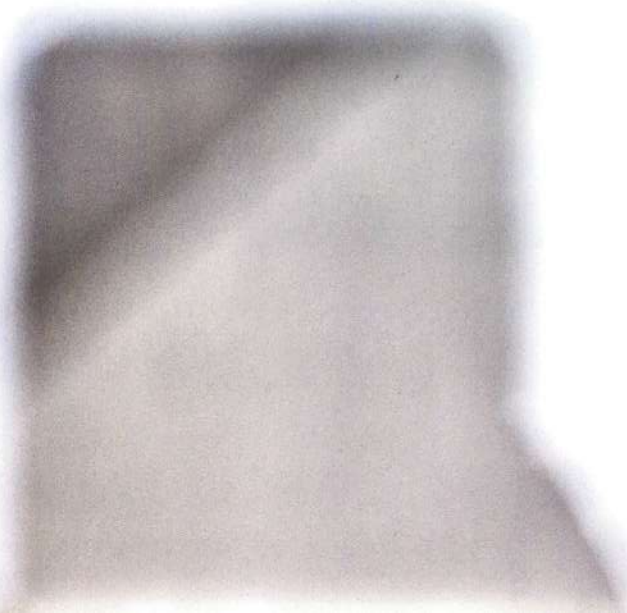
Program 8

```
create or replace trigger val_stock
before insert on order_items_table
for each row
declare
        r_avail_stock Number;
        r_pending_orders Number;
begin
        select stock_level into r_avail_stock from
            inv_table where item_id = new.item_id;

if r_avail_stock < new.quantity then
    raise_app_error;
end if;

select sum(quantity) into r_pending_orders from
    orders_items_table oi join order_table on
    oi.order_id = o.order_id where oi.item_id =
    new.item_id and o.order_status = 'pending';

if r_available_stock - r_pending orders <
                                new.quantity
                                then
    raise_app_error
end if;
end;
```



Ex.No.: 14	MONGO DB
Date: 8/10/24	

MongoDB is a free and open-source cross-platform document-oriented database. Classified as a NoSQL database, MongoDB avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster.

Create Database using mongosh

After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.

If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the myFirstDatabase database.

Show all databases

To see all available databases, in your terminal type show dbs.

Notice that myFirstDatabase is not listed. This is because the database is empty. An empty database is essentially non-existant.

Change or Create a Database

You can change or create a new database by typing use then the name of the database.

Create Collection using mongosh

You can create a collection using the createCollection() database method.

Insert Documents

insertOne()

```
db.posts.insertOne({
 title: "Post Title 1",
 body: "Body of post.",
 category: "News",
 likes: 1,
 tags: ["news", "events"],
```

date: Date()

})

EXERCISE 18

Structure of 'restaurants' collection:


```
{
 "address": {
   "building": "1007",
   "coord": [ -73.856077, 40.848447 ],
   "street": "Morris Park Ave",
   "zipcode": "10462"
 },
 "borough": "Bronx",
 "cuisine": "Bakery",

 "grades": [
   { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
   { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
   { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
   { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
   { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
 ],
 "name": "Morris Park Bake Shop",
 "restaurant_id": "30075445"

}
```


1. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which prepared dish except 'American' and 'Chinees' or restaurant's name begins with letter 'Wil'.

```
db.restaurants.find({$or: [{cuisine:{$nin: ['American ', 'Chinese']}}, {name: {$regex: 'Wil', $options: 'i'}}]},
{_id:1, name: 1, borough: 1, cuisine: 1});
```

2. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates..

```
db.restaurants.find({"grades.grade": "A", "grades.score": 11,
"grades.date": new ISODate("2014-08-11")},
{_id:1, name:1, grades: 1});
```

3. Write a MongoDB query to find the restaurant Id, name and grades for those restaurants where the 2nd element of grades array contains a grade of "A" and score 9 on an ISODate "2014-08-11T00:00:00Z".

```
db.restaurants.find({"grades.1.grade": "A",
"grades.1.score": 9, "grades.1.date": new isodate("2014-08-11")
}, {_id:1, name:1, grades:1});
```

4. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of coord array contains a value which is more than 42 and upto 52..

```
db.restaurants.find({"address.coord.1": {$gt:42, $lte:52
}}, {_id:1, name:1, address:1, "address.coord":1});
```

5. Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

```
db.restaurants.find().sort({name:1});
```

6. Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

```
db.restaurants.find().sort({name:-1});
```

7. Write a MongoDB query to arranged the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
db.restaurants.find().sort({cuisine:1, borough:-1});
```

8. Write a MongoDB query to know whether all the addresses contains the street or not.

```
db.restaurants.find({"address.street": {$exists:true,
$ne:""}});
```

9. Write a MongoDB query which will select all documents in the restaurants collection where the coord field value is Double.

```
db.restaurants.find({"address.coord": {$type: "double"}});
```

10. Write a MongoDB query which will select the restaurant Id, name and grades for those restaurants which returns 0 as a remainder after dividing the score by 7.

```
db.restaurants.find({"grades.score": {$mod: [7, 0]}}, {_id:1, name:1, grades:1});
```

11. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'mon' as three letters somewhere in its name.

```
db.restaurants.find({name: {$regex: "mon", $options: "i"}}, {_id:1, name:1, borough:1, "address.coord":1, cuisine:1});
```

12. Write a MongoDB query to find the restaurant name, borough, longitude and latitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.

```
db.restaurants.find({name: {$regex: "^Mad", $options: "i"}}, {_id:1, name:1, borough:1, "address.coord":1, cuisine:1});
```

13. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5.

```
db.restaurants.find({"grades.score": {$lt: 5}});
```

14. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan.

```
db.restaurants.find({"grades.score": {$lt:5}, borough: "Manhattan"});
```

15. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn.

```
db.restaurants.find({"grades.score": {$lt: 5},
borough: {$in: ["Manhattan", "Brooklyn"]}
});
```

16. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
db.restaurants.find({"grades.score": {$lt: 5},
borough: {$in: ["Manhattan", "Brooklyn"]},
cuisine: {$ne: "American"}
});
```

17. Write a MongoDB query to find the restaurants that have at least one grade with a score of less than 5 and that are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
db.restaurants.find({"grades.score": {$lt: 5},
borough: {$in: ["Manhattan", "Brooklyn"]},
cuisine: {$in: ["American", "Chinese"]}
});
```

18. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6.

```
db.restaurants.find({"grades.score": {$in: [2, 6]}
});
```

19. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan.

```
db.restaurants.find({"grades.score": {$in: [2, 6]},
borough: "Manhattan"});
```

20. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn.

```
db.restaurants.find({"grades.score": {$in: [2, 6]},
borough: {$in: ["Manhattan", "Brooklyn"]}});
```

21. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American.

```
db.restaurants.find({"grades.score": {$in: [2, 6]},
borough: {$in: ["Manhattan", "Brooklyn"]},
cuisine: {$ne: "American"}});
```

22. Write a MongoDB query to find the restaurants that have a grade with a score of 2 and a grade with a score of 6 and are located in the borough of Manhattan or Brooklyn, and their cuisine is not American or Chinese.

```
db.restaurants.find({"grades.score": {$in: [2, 6]},
borough: {$in: ["Manhattan", "Brooklyn"]},
cuisine: {$in: ["American", "Chinese"]}});
```

23. Write a MongoDB query to find the restaurants that have a grade with a score of 2 or a grade with a score of 6.

```
db.restaurants.find({"grades.score": {$in: [2, 6]}});
```

Sample document of 'movies' collection

```
{
  _id: ObjectId("573a1390f29313caabcd42e8"),
plot: 'A group of bandits stage a brazen train hold-up, only to find a determined posse hot on
their heels.',
genres: [ 'Short', 'Western' ],
runtime: 11,
cast: [
   'A.C. Abadie',
   "Gilbert M. 'Broncho Billy' Anderson",
   'George Barnes',
   'Justus D. Barnes'
 ],
```

poster: 'https://m.media-amazon.com/images/M/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYjgtMmYwYWIxZDYyNzU2XkEyXkFqcGdeQXVyNzQzNzQxNzI@._V1_SY1000_SX677_AL_.jpg',
title: 'The Great Train Robbery',

fullplot: "Among the earliest existing films in American cinema - notable as the first film that presented a narrative story to tell - it depicts a group of cowboy outlaws who hold up a train and rob the passengers. They are then pursued by a Sheriff's posse. Several scenes have color included - all hand tinted.",

languages: ['English'],
released: ISODate("1903-12-01T00:00:00.000Z"),
directors: ['Edwin S. Porter'],
rated: 'TV-G',
awards: { wins: 1, nominations: 0, text: '1 win.' },
lastupdated: '2015-08-13 00:27:59.177000000',
year: 1903,
imdb: { rating: 7.4, votes: 9847, id: 439 },
countries: ['USA'],
type: 'movie',
tomatoes: {
viewer: { rating: 3.7, numReviews: 2559, meter: 75 },
fresh: 6,
critic: { rating: 7.6, numReviews: 6, meter: 100 },
rotten: 0,
lastUpdated: ISODate("2015-08-08T19:16:10.000Z")
}

1. Find all movies with full information from the 'movies' collection that released in the year 1893.

db.movies.find({released: { $gte: new Date("1893-01-01"), $lt: new Date("1894-01-01")}})

2. Find all movies with full information from the 'movies' collection that have a runtime greater than 120 minutes.

db.movies.find({runtime: {$gt: 120}})

3. Find all movies with full information from the 'movies' collection that have "Short" genre.

db.movies.find({genres: "Short"})

4. Retrieve all movies from the 'movies' collection that were directed by "William K.L. Dickson" and include complete information for each movie.

db.movies.find({directors: "William K.L. Dickson"})

6. Retrieve all movies from the 'movies' collection that were released in the USA and include complete information for each movie.

db.movies.find({countries: "USA"})

7. Retrieve all movies from the 'movies' collection that have complete information and are rated as "UNRATED".

db.movies.find({rating: "UNRATED"})

8. Retrieve all movies from the 'movies' collection that have complete information and have received more than 1000 votes on IMDb.

```
db.movies.find({imdbVotes: {$gt:10003}})
```

9. Retrieve all movies from the 'movies' collection that have complete information and have an IMDb rating higher than 7.

```
db.movies.find({imdbRating: {$gt:7}})
```

10. Retrieve all movies from the 'movies' collection that have complete information and have a viewer rating higher than 4 on Tomatoes.

```
db.movies.find({tomatoes.viewer.rating: {$gt:4}})
```

11. Retrieve all movies from the 'movies' collection that have received an award.

```
db.movies.find({awards: {$exists: true, $ne: []}})
```

12. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB that have at least one nomination.

```
db.movies.find({nominations: {$exists: true,
  $ne: []}},
 {title:1, language:1, released:1, directors:1,
  writers:1, awards:1, year:1, genres:1, runtime:1,
  cast:1, countries:1})
```

13. Find all movies with title, languages, released, directors, writers, awards, year, genres, runtime, cast, countries from the 'movies' collection in MongoDB with cast including "Charles Kayser".

```
db.movies.find({cast: "Charles Kayser"},
{title:1, languages:1, released:1, directors:1,
writers:1, awards:1, year:1, genres:1, runtime:1
cast:1, countries:1})
```

14. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that released on May 9, 1893.

```
db.movies.find({released: new Date("1893-05-09")},
{title:1, languages:1, released:1, directors:1,
writers:1, countries:1})
```

14. Retrieve all movies with title, languages, released, directors, writers, countries from the 'movies' collection in MongoDB that have a word "scene" in the title.

```
db.movies.find({title: /scene/i}, {title:1,
languages:1, released:1, directors:1, writers:1})
```

Evaluation Procedure	Marks awarded
PL/SQL Procedure(5)	
Program/Execution (5)	
Viva(5)	
Total (15)	
Faculty Signature	

Ex.No.: 15		OTHER DATABASE OBJECTS
Date:	18/10/24	

OTHER DATABASE OBJECTS

Objectives

After the completion of this exercise, the students will be able to do the following:
- Create, maintain, and use sequences
- Create and maintain indexes

Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.
If you want to improve the performance of some queries, you should consider creating an index. You
can also use indexes to enforce uniqueness on a column or a collection of columns.
You can provide alternative names for objects by using synonyms.

What Is a Sequence?
A sequence:
- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

CREATE SEQUENCE *sequence*
[INCREMENT BY *n*]
[START WITH *n*]
[{MAXVALUE *n* | NOMAXVALUE}]
[{MINVALUE *n* | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE *n* | NOCACHE}];
In the syntax:
sequence is the name of the sequence generator

1) create sequence dept_id_seq start with 200 increment by 10 maxvalue 1000 nocycle;

2) select sequence_name, max_value, increment_by, last_number from all_sequences where sequence_name = 'Dept_id_seq';

3) insert into dept (dept_id, dept_name) values (dept_id_seq.nextval, 'Education');

insert into dept(dept_id, dept_name) values (dept_id_seq.nextval, 'Administration');

commit;

select * from dept where dept_name in ('Education', 'Administration');

4) create index emp_dept_id_idx on emp(dept_id);

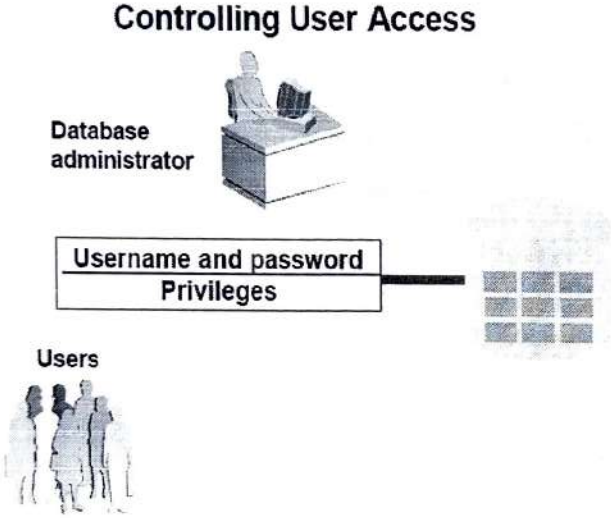
5) select index_name, uniqueness from all_indexes where table_name = 'EMP';

Ex.No.: 16		CONTROLLING USER ACCESS
Date:	25/10/24	

Objectives

After the completion of this exercise, the students will be able to do the following:

- Create users
- Create roles to ease setup and maintenance of the security model
- Use the GRANT and REVOKE statements to grant and revoke object privileges
- Create and access database links



Controlling User Access

In a multiple-user environment, you want to maintain security of the database access and use. With Oracle server database security, you can do the following:

- Control database access
- Give access to specific objects in the database
- Confirm given and received *privileges* with the Oracle data dictionary
- Create synonyms for database objects

Privileges

- Database security:
  - System security
  - Data security

Find the Solution for the following:

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?

"create session" allows the user to log in to the oracle database

2. What privilege should a user be given to create tables?

"create table" allows the users to create tables within their schema

3. If you create a table, who can pass along privileges to other users on your table?

The owner of the table can pass along privileges to other users on their table

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

A role is a collection of privileges that can be granted to users.

5. What command do you use to change your password?

alter user <username> identified by <new password>;

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

7. Query all the rows in your DEPARTMENTS table.

8. 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.

9. Query the USER_TABLES data dictionary to see information about the tables that you own.

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

11. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

6) grant select on departments to <other_user>;
grant select on <other_user>.departments to <your-username>;

7) select * from departments;

8) Insert into departments (dept_id, dept_name) values (500, 'Education');
Insert into departments (dept_id, dept_name) values (510, 'Human Resources');
select * from departments where dept_id in (500, 510);

9) select * from user_tables;

10) revoke select on departments from <other_user>;

11) delete from departments where dept_id=500;
commit;