# RAJALAKSHMI ENGINEERING COLLEGE

**RAJALAKSHMI NAGAR, THANDALAM – 602 105**

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

## CS23332 DATABASE MANAGEMENT SYSTEMS LAB

**Laboratory Record Note Book**

Name : Meghna Rajesh

Year / Branch / Section : II AIML-B

University Register No. : 211623160 1098

College Roll No. : 231501098

Semester : III

Academic Year : 2023 - 2024

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

# BONAFIDE CERTIFICATE

NAME Meghna Rajesh Mellam

ACADEMIC YEAR 2023-2027 SEMESTER III BRANCH AIML

UNIVERSITY REGISTER No. 2116231501098

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

DataBase Management Systems (CS23332) Laboratory during the year **2024 - 2025**

**Signature of Faculty - in - Charge**

**Submitted for the Practical Examination held on** 23/11/2024

**External Examiner**

**Internal Examiner**

# CS23332 DATABASE MANAGEMENT SYSTEMS

| | |
|---|---|
| **NAME** | Meghna Rajesh |
| **ROLL NO** | 231501098 |
| **DEPT** | AIML |
| **SEC** | B |

# INDEX PAGE

| DATE | SL.NO | NAME OF THE EXPERIMENT | MARKS | SIGNATURE OF THE FACULTY |
|---|---|---|---|---|
| 26/7/24 | 01 | CREATION OF BASE TABLE AND DML OPERATIONS | 10 | |
| 30/7/24 | 02 | DATA MANIPULATIONS | 10 | |
| 2/8/24 | 03 | WRITING BASIC SQL SELECT STATEMENT | 10 | |
| 6/8/24 | 04 | WORKING WITH CONSTRAINTS | 10 | |
| 13/8/24 | 05 | CREATING VIEWS | 10 | |
| 20/8/24 | 06 | RESTRICTING AND SORTING DATA | 10 | |
| 27/8/24 | 07 | USING SET OPERATORS | 10 | |
| 3/9/24 | 08 | WORKING WITH MULTIPLE TABLES | 10 | |
| 16/9/24 | 09 | SUB QUERIES | 10 | |
| 20/9/24 | 10 | AGGREGATING DATA USING GROUP FUNCTIONS | 10 | |
| 24/9/24 | 11 | PL SQL PROGRAMS | 9 | |
| 27/9/24 | 12 | WORKING WITH CURSOR PROCEDURES AND FUNCTIONS | 9 | |
| 8/10/24 | 13 | WORKING WITH TRIGGER | 9 | |
| 8/10/24 | 14 | MONGO DB | 9 | |
| 18/10/24 | 15 | OTHER DATABASE OBJECTS | 10 | |
| 29/10/24 | 16 | CONTROLLING USER ACCESS | 10 | |

Complete

| 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>;

**STEP-8:** The truncate command deletes all rows from the table. Only the structure of the table remains.

Syntax:

TRUNCATE TABLE <table name>;

**STEP-9:** Alter the existing table using ALTER statement.

Syntax:

Add Column:

ALTER TABLE <table name> ADD (column data type [DEFAULTexpr][,column data type]);

Modify Column:

ALTER TABLE <table name> MODIFY (column data type [DEFAULT expr], [,column data type]);

Drop Column:

ALTER TABLE <table name> DROP COLUMN <column name>;

**STEP-10:** To drop the entire table using DROP statement.

Syntax:

DROP TABLE <table name>;

**STEP-11:** Exit.

1. Create MY_EMPLOYEE table with the following structure

Create table my-employee (id number (4), last_name varchar(25), first_name varchar(25), user id varchar(8), salary number(9,2)),

| NAME | NULL? | TYPE |
|---|---|---|
| ID | Not null | Number(4) |
| Last_name | | Varchar(25) |
| First_name | | Varchar(25) |
| Userid | | Varchar(25) |
| Salary | | Number(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 (1, 'patel', 'Ralph', 'rpatel', 895);
Insert into my-employee values (2, 'Dones', 'Betty', 'balances, 860);

3. Display the table with values.

select * from my-employee.

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 (3 'Biri', 'Ben', 'bbiri', 1100);
Insert into my-employee (4, 'Newman', 'Chad', 'newman, 750);
update my-employee set userid, concat ( first_name ), (last_name
where ID=3

5. Delete Betty dancs from MY _EMPLOYEE table.

Delete from my-employee where first_name =
'Betty' and last_name = 'Dones';

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 = 'Dreller'
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: | 30/7/24 | |

**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 emp_id , first_name, last_name , salary from
                                        employee;
```

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

```
Select emp_id , first_name , last_name , salary from employee
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 employee where salary>4800;
```

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

```
Select *from employee where last_name = 'Rishi'
```

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

```
Select first_name , last_name from employee where
dept_id Int (60,90,80);
```

(f ) Display the unique Manager_Id.

```
Select distinct manager_id from employee;
```

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(5) primary key , empname
varchar(50), job varchar(50), Basic decimal(10,2) , DA
decimal(10,2) , PF decimal(10,2) , netpay decimal(10,2)),.
```

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

```
Insert into employees values ('1', Rishi , 5000, 'Manager',
50000*030, 50000*040 ; 50000+0·12(50000+150000*0·30)
+ (50000+0·401)
```

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

```
Select empno, empname , job Basic from emp where
Basic = select min (Basic) from emp
```

(c) If Net Pay is less than

```
Select empno , empname , netpay from emp where netpay <30000;
```

DEPARTMENT TABLE

Create table dept(dept-id number(6) not null, dept name varchar (20) not null, manager_id number(6) location_id number(4));

JOB CREATE TABLE:

Create table jobs-grade (Grade_level varchar(2)); lowest-sal number(25), highest-sal number(6));

LOCATION TABLE:

Create table location(location-id number(4) not null, st-addr varchar(40), postal-code varchar(2), city varchar(30) not null, state-province varchar(25), country id. char(2));

**DEPARTMENT TABLE**

| NAME | NULL? | TYPE |
|---|---|---|
| Dept_id | Not null | Number(6) |
| Dept_name | Not null | Varchar(20) |
| Manager_id | | Number(6) |
| Location_id | | Number(4) |

**JOB_GRADE TABLE**

| NAME | NULL? | TYPE |
|---|---|---|
| Grade_level | | Varchar(2) |
| Lowest_sal | | Number |
| Highest_sal | | Number |

**LOCATION TABLE**

| NAME | NULL? | TYPE |
|---|---|---|
| Location_id | Not null | Number(4) |
| St_addr | | Varchar(40) |
| Postal_code | | Varchar(12) |
| City | Not null | Varchar(30) |
| State_province | | Varchar(25) |
| Country_id | | Char(2) |

1. Create the DEPT table based on the DEPARTMENT following the table instance chart below. Confirm that the table is created.

| Column name | ID | NAME |
|---|---|---|
| Key Type | | |
| Nulls/Unique | | |
| FK table | | |
| FK column | | |
| Data Type | Number | Varchar2 |
| Length | 7 | 25 |

Create table dept (id-no number (7), dept_name varchar(25), Primary (id-no));

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 (not null, last-name varchar (25), first-name varchar 2(25), dept-id number);
desc emp;

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));
desc emp;

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 2 as select ID as id,
first-name as first-name;
last-name as last-name;
salary as salary;
Dept-id as Dept-id from emp.

5 Drop the EMP table.

Drop table emp;

6 Rename the EMPLOYEES2 table as EMP.

alter table employee 81 rename to EMP;

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

Comment on table dept is table containing dept details,
Commant on table emp is Table containing emp details;

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

Alter table emp drop column first-name;
desc emp;

| 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 JOB" 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;

select employee-id, last-name, salary * 12
Annual-salary from employee;

**Queries**

Select employee-id, last_name, sal * 12 as ANNUAL SALARY from employees.

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

desc department,
select * from department,

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 number, last-name, jobs.code, hire-date from employee e;

4. Provide an alias STARTDATE for the hire date.

Select employee-number, last-name, job-code, hire-date as start data from employees;

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

select distinct job-code from employee;

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

Select concat (class-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 concat-ws(',', employee-number, last-name, job-code, 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 | |

Select employee-id || " || first-name || ", ||last-name|| ", " job-id as output from employee;

| 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-pic
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 and constraint my-dept-id-plc
Primary key (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;
alter table emp add constraint my-emp-dept-id
foreign key (Dept-id)
reference 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) 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 emp-id; first_name employee, dept-id from employees.

2. Display the contents of the EMPLOYEES_VU view.

select * from employee.

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

select view_name, text from user_views

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

select employee, dept-id from employees.

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 dept 50 as select emp-id, first_name, dept-id from employee where dept-id = 50.

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

Select * from dept 50

7. Attempt to reassign Matos to department 80.

update employee-vu set employee = 'Matos' where dept_id = 80;

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 (last name, department_name, salary, salary grade) as select employee · Clast nam, department name, salaries, job grade, salary from employees, job- grade.

| 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 employee where
salary > 12000;
```

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

```
Select last_name, department_number from employees
where employee_number = 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, start_date from employees
where start_date between '1998-02-20' and '1998-05-01'
Order by start_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_number from emp where
dept_number IN (20,50) order by last_name;
```

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 emp, salary As "monthly_salary"
from employees where salary btw 5000 & 12000
and dept_number IN (20,50) order by last_name;
```

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

```
Select last_name, As employee, 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_title 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_pct from
emp where commission_pct is not null order by
salary desc ,commission_pct desc;
```

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

```
Select last_name from emp 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%e%' or last_name like '%e%a%';
```

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 emp where job_id
IN ('SA_REP', 'ST_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: | 27/9/24 | |

**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.

1. Select department_id
From employee
Minus
Select department_id
From employee
where upper(job_id) = upper('ST_CLERK')
order by;

2. Select country_id, country_name
Minus
Select country_id, country_name
From countries
join locations
using (country_id)
join department

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.

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.

```
using (location_id)
where department_id is not null.
```

3. 
```
Select distinct job_id, department_id
From employee
where department_id = 10
union all
select distinct job_id, department_id
From employee
whered department_id = 50
union all
Select distinct job_id, department_id
From employee
where department_id = 20
```

4. 
```
Select employee_id, job_id
From employee
Intersect
Select emp_id, job_id
From job_history
order by 1;
```

5. 
```
Select last_name, department_id, To_char('null')
From employee
union
Select to_char('null'), dept_id, dept_name
From department
order by;
```

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

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).

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 | |

| 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 employee e, dept d.
Where e. department_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 jds_id, location_id
From employee, dept
Where employees, dept_id= dept. dept_id and employees.
   depts 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, d.location_id , l.city
From employees e.depart d, location L
Where e.dept_id=d.dept_id
And
d. location_id = l. location_id.
And e.commission_put is not null;
```

4. Select last_name , department_name
From employee, departments
Where employees.dept_id = dept.dptss_id
And last_name like '%a%';

5. Select e.last_name, e.job_id , e.dept_id, d.dept_name
From employee e. joind dept d
on ( e.dept_id= d.dept_id)
join locations l
on ( d location_id = l. location_id)
where lower (l.city) = 'tronto';

9. desc job_grades
Select e.last_name, e.job_id, d.dept_name
e salary, j, grade_level;
From employee, dept_d, job_grades
where e.dept_id = d.dept_id
And e.salary Between j.lowest_sal and j.highest_sal;

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

5. Write a query to display the last name, job, department number, and department name for all employees who work in 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 w.last_name "Employees". w.employee_id
                                    "EMP#",
    m.last_name "Manager". m.employee_id "Mgr#":
From employee w join employees m
on (w.manager_id = m.emp_id).
```

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

```
Select w.last_name "Employee". w.employee_id = "EMP#",
    m.last_name "Manger", m.emp_id "MGR#",
From employee w
Left outer join employee m
    (w.manager = m.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 e.dept_id.dept, e.last_name employee
    c.last_name colleague.
From employee e join employee c
on (e.dept_id = c.dept_id)
where e.employee_id <> c.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

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 employee e join employee davies
on (davis, last_name = 'davies')
where davies.hire_date < e.hire_date
```

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 w.last_name, w.hire_date, m.last_name, m.hire_date
From employee w join employee m
on (w.manager_id = m.employee_id)
where w.hire_date < m.hire_date);
```

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

| Ex.No.: 9 | | SUB QUERIES |
|---|---|---|
| Date: | 16/9/24 | |

**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).

```
Select last_name, hire_date from employees
where dept_id = (select dept_id from emp)
 where last_name like 'name') and last_name <>
    & '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 employee
where salary > (select avg (salary) from emp)
 order by salary;
```

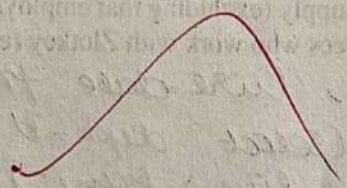
3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a *u*.

```
Select employee_id, last_name
From employees
where department_id (select dept_id from emp where
last_name like '%u%');
```

4. Select last_name, department_id, job_id, From employee where dept_id in (select dept_id from department where location_id = & location);

Modify the query so that the user is prompted for a location id

Select last_name, dept_id, job_id from employee where dept_id in (select dept_id from depts where location_id = & location);



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.

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

```
Select last_name, salary
From employees
where manager_id in (select emp_id from
                     employee where 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 department_id, last_name, job_id from
where dept_id in (select dept_id from depts
  where 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 emp_id, last_name, salary from employee
where salary > (select avg (salary) from emp)
  and dept_id in (select dept_id from
      employee where last_name like '%u%');
```

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

| 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

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

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

**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 max (salary) as maximum, avg (salary) As Average,
Sum (salary) as total min (salary as minimum from
emp
```

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

```
Select current_job_id, max (salary) as maximum,
Round Avg(salary) as Average, sum (salary) as
total, min (salary) As minimum
From emp
Group by current_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.

```
Select current_jds_id count ( current_jds_id) as
jobs from employee
Group by current_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 (current_job_id) as "Number of managers
from employees
    where current_jds_id like '%.MGR%' or current jds_id
                                    like '%.MGR%';
```

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

```
Select max(salary) - min(salary) as difference
From employee.
```

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 reports to, min (salary)
from emp;
where Salary <= 6000 and reports_to is not null
Group by report_to
order by min (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 'total':
sum (if (hire_date like '1995%.' ,1,0)) as '1995';
sum ( if (hire_date like '1997%.' ,1)) as '1997';
sum (if (hire_date like '1998%.';1,0)) as '1998',
from employee;
```

11) SELECT JOB TYPE as Job;

SUM(case WHEN dept_ID = 20 then Salary else 0 end) as Salary-dept-20,

SUM (case WHEN dept_ID = 50 then Salary else 0 end) as Salary-dept-80,

SUM (case WHEN dept_ID = 90 then Salary else 0 and) as Salary-dept-90,

SUM (Salary) as total salary FROM employees

WHERE department_id in (20,50,80,90) GROUP BY JOB_ID;

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 dept_name as "Department", location_ID as "location", count(employee_ID) as "Number of people", round(avg(salary,2)) as "Salary" FROM employees GROUP BY dept_name, location;

| Evaluation Procedure | Marks awarded |
|---|---|
| Query(5) | 5 |
| Execution (5) | 5 |
| Viva(5) | 5 |
| Total (15) | 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:

```
DECLARE
emp_id employee.emp_id %TYPE:=110;
emp_name employees.name%TYPE;
emp_salary employees.salary%TYPE;
incentive NUMBER(7,2);

BEGIN
SELECT name, salary
INTO emp_name, emp_salary
FROM employees
WHERE emp_id=110;
incentive := emp_salary * 0.10;

DBMS_OUTPUT.PUT_LINE('Employee name: '||emp_name);
DBMS_OUTPUT.PUT_LINE('Employee salary: '||emp_salary);
DBMS_OUTPUT_PUT_LINE('Incentive (10%): '||incentive);

EXCEPTION:
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee with ID 110 not found')
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: '||SQL);
END;
```

PROGRAM 1

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

PROGRAM 2

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

```
SET SEVEROUTPUT ON;
DECLARE
    employee VARCHAR(50) = 'John Doe';
  "Employee" VARCHAR(50) = 'Jane Doe';
BEGIN
   DBMS_OUTPUT.PUT_LINE ('case_ Insensitive (employee Name');
DBMS_OUTPUT.PUT_LINE ("Case-Sensitive ('employee name');
EXCEPTION
        DBMS_OUTPUT.PUT_LINE ('Error: ' || SQLERRM);
END;
```

PROGRAM 3

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

```
SET SERVER OUTPUT ON;
BEGIN
   UPDATE employees
    SET salary = salary + (salary * 0.10)
   WHERE emp_id = 12
   RETURNING salary INTO :new_salary;
   DBMS_OUTPUT.PUT_LINE ('New Salary : ' || :new_salary);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employee with ID122 not found');
    WHEN OTHER THEN
        DBMS_OUTPUT.PUT_LINE ('ETTOR : ' || SQLERRM)
   END;
```

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.

```
SET SEVER OUTPUT. ON;
BEGIN
    IF('Hello' IS not null AND null IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('Both are not null');
ELSE
    DBMS_OOTPUT.PUT_LINE('Atleast one is null');
END IF
END;
```

Output: Atleast one is NULL

PROGRAM 5

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

Output: Pattern 1 matched
Pattern 2 matched
Pattern 3 matched

```
SET SEVER OUTPUT ON;
BEGIN
    IF 'Hello world' LIKE 'H%. W%' THEN
    DBMS_OUTPUT. PUT_LINE ('Patter 1 matched)
    END IF;
  IF 'Hello 123' LIKE 'HELLO_23' THEN
      DBMS_OUTPUT. PUT_LINE ('Pattern 2 matched)
    END IF;
IF '50% discount' LIKE '50 %:%' ESCAPE '\' THEN
   DBMS_OUTPUT . PUT_LINE ('Pattern 3 matched with escape.');
```

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.

Output: small: 10
large: 20

```
SET SEVEROUTPUT ON;
DECLARE
    num1 NUMBER = 10;
    num2 NUMBER = 20;
    num_small NUMBER := LEAST (num1, num2);
    num_large NUMBER := GREATEST (num1, num2);
BEGIN
    DBMS_OUTPUT. PUT_LINE ('Small: ' || num_small ||, 'Large. ' ||
    END;
```

**PROGRAM 7**

**Write a PL/SQL procedure to calculate the incentive on a target achieved and display the message either the record updated or not.**

```
SET SEVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE calc_incentive (emp_id IN number) IS
BEGIN
UPDATE employees SET incentive = target_achieved * 0.10 WHERE
                                                    emp_id AND TARGET.

DBMS_OUTPUT.PUT_LINE('Record '||case WHEN SQL %ROW COUNT > 0
                                                    THEN 'UPDATED.
ELSE 'not UPDATED.' END);

END;
```

**PROGRAM 8**

**Write a PL/SQL procedure to calculate incentive achieved according to the specific sale limit.**

```
SET SEVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE calc_incentive (emp_id IN number)
 IS sales_limit NUMBER = 1000;
 incentive
BEGIN
   SELECT CASE WHEN total_sales >= sales_limit THEN total_sales
                                                    *0.10
 UPDATE employees SET incentive = incentive_amount WHERE
                                        emp_id = emp_id;
 DBMS_OUTPUT.PUT_LINE ('Incentive for ID' || emp_id || ':' ||
                                                    incentive_amt);
 EXCEPTION:
    WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE

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.**

```
SET SEVEROUTPUT ON;
DECLARE
    emp_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO emp_count FROM employees WHERE
                                          dept_id=50;
  DBMS_OUTPUT.PUT_LINE('employees in DEPT50: '||emp_count);
  DBMS_OUTPUT.PUT_LINE(IIF(emp_count < 45,'vacancies available'),
  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.**

```
SET SEVEROUTPUT ON;
  DECLARE
      emp_count number;
      vacancies number=45;
  Begin
    Select count(*) INTO emp_count from employees
                                  where dept=50
DBMS_output.put_line('Employees in Dept 50: '||emp_count||
                                          Vaccancies)),
  END;
```

PROGRAM 11

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

```
Set serveroutput on;
Begin
For rec in (select employee_id, name, job_title, hire_date,
                                        Salary FROM DBMS_output
        ', name : ' || rec_name ||
        ', Job title: ' || rec.job_title ||
        ', Hire DATE: ' || re.hire_date ||
', Salary: ' || rec.salary);
END LOOP;
END;
```

PROGRAM 12

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

```
Set sever output ON;
Begin
For rec in (select e.employee_id, e.name, d.dept_name
        FROM employees e
        JOIN dept d on e.dept_id = d.dept
DBMS_output.putline (ID: || rec.employees_id||
        ', name : ' || rec.name ||
        ', Department : ' || ree.dept_name );
END LOOP;
END;
```

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

```
SET Server output ON;
Begin
  For rec in (select job id, job_title, min_salary From j)
    LOOP
  DBMS_output.put_line ('Job ID : ' ||rec.job id ||
                        ', Title: ' ||re.job title ||
                        ', min salary : ' rec.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.

```
Set serveroutput on;
  Begin
    For rec (select e.employee_id, e.name, j.start_date
             From employees e
             Join job_history j on e.employee_id = j.emp_id
  DBMS_output.put_line ('ID: ' || rec.emp_id ||
                        ', name: ' || rec.name ||
                        ', Job start Date: ' || rec.start_date);
  END LOOP;
  END;
```

PROGRAM 15

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

```
Set serveroutput on;
Begin
  For rec in (select e.employee-id, e.name, j.end-date
        From employeese.
    JOIN job_history j on e.emp_id=j.employee_id)
DBMS-output.put_line ('ID:' ||rec.employee_id||
                      ', Name:' ||rec.name||
                      ', Job End Date:' rec.end-date);
  END LOOP
END;
```

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

| Ex.No.: 12 | | WORKING WITH CURSOR, PROCEDURES AND FUNCTIONS |
|---|---|---|
| Date: | 27/09/24 | |

**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**

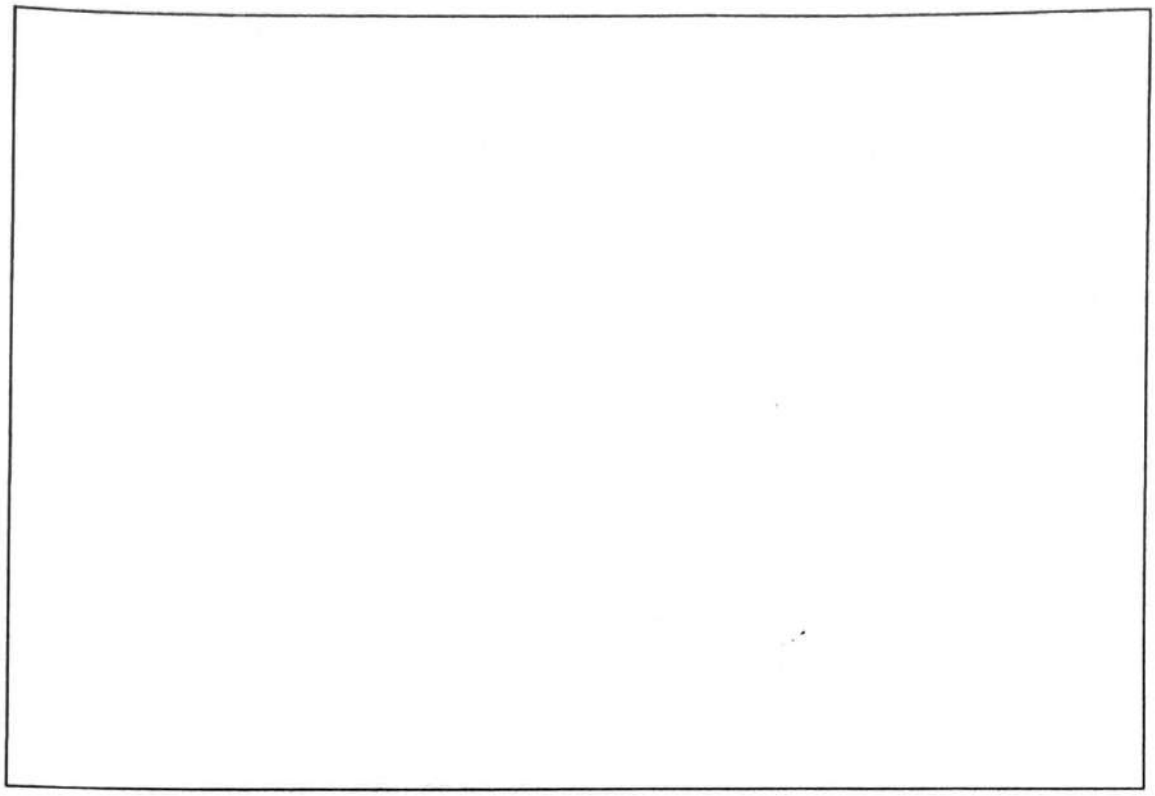
```
Create or replace function factorial (n IN number)
 return number IS
result number := 1;
Begin
   If n<0 Then
    Return null;
  Elsif n=0 OR n=1 then
   return 1;
   Else
     for i in 2. n loop
   Result := result i=;
   End loop;
   End if;
Return result;
EXCEPTION
    when Others then
DBMS_ output_line ('An Error occured '|| SQLERRM);
 Return null;
End factorial
Set serveroutput on;
Declare
    num number:=5;
    fact number;
End;
```

Program 2:

```
Create Table Books(
    book_id number(5) Primary key, title varchar2(100),
    author varchar(100), Publication_year number(4),
    available_copies number(5));

Insert into books values (1, '1984', 'orwell', 1949, 4)
Insert into books values (2, 'Mocking bird', 'lee', 1960, 2);
Insert into books values(3, 'Gatsby', 'Fitzball', 1925, 5);

Commit;
Create or replace prodedure getbook (P_book_id in
  number, P_title out varchar2, P_author out varchar2)
Select title, author into P_title, P_author from Books
  where_id
End;

Set server output on;
Declare
   v_title varchar (100), v_author varchar2(100)
Begin
   Getbook in tonyId (1, v_title, v_author);
DBMS_output, Put_line ('Book: ' || v_title || "Author:' ||
                                        v_author)
END;.
```

Program 2

**Write a PL/SQL program using Procedures IN,INOUT,OUT parameters to retrieve the corresponding book information in library**



**TO WRITE A PL/SQL BLOCK TO DISPLAY THE EMPLOYEE ID AND EMPLOYEE NAME WHERE DEPARTMENT NUMBER IS 11 USING EXPLICIT CURSORS**

```
1  declare
2  cursor cenl is select eid,sal from ssempp where dno=11;
3  ecode ssempp.eid%type;
4  esal empp.sal%type;
5  begin
6  open cenl;
7  loop
8  fetch cenl into ecode,esal;
9  exit when cenl%notfound;
10 dbms_output.put_line(' Employee code and employee salary are' || ecode 'and'|| esal);
```

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

| Ex.No.: 13 | | WORKING WITH TRIGGER |
|---|---|---|
| Date: | 8/10/24 | **TRIGGER** |

## 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
For each row
Declare
    child_count number;
Begin
 Select count(*) into child_count From child where
 Parent_id if child_count >0  Raise_application_error.
 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 table sampleTable(
            id number(5) Primary key,
            name varchar(50) NOT,
            email varchar2(100) unique );

Create or replace trigger check_duplicate_email
Before insert or update on sample table.
Declare
      duplicate_count number*
Begin
   Select count(*) into duplicate_count
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_total_sales.
Before insert on Sales
For Each row
 Begin
   if (select sum(amount) fromm Sales): new amount>5000
   raise_application_error(-20002), 'Total exceeds threshold')
 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 Log_Salary_Changes
After update of Salary on employees
For each row
Begin
Insert into Employeeaa  VALUES(audit_seq.NEXTVAL, :OLD
emp_id : OLD: salary, : new·salary, SYSDATE);
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 record_user_activity
After insert or update or Delete on Employees for
each row
Begin
  Insert into Audit log values (audit_seq.NEXTVAL,
   Case when .inserting then 'INSERT' when updating
   then 'UPDATE'
'Employees', NVL (:OLD.emp_id , :New emp id), SYSDATE, USER);
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 table sales (
          Sales_id number Primary key, amount
          number(10,2), running_total number(10,2)
          );
Create or replace trigger update_running total
for each row
Begin
  Select NVL(max(running_total,0)+ :New amount
  into :new.running
END;
```

Program 8

Write a code in PL/SQL to create a trigger that validates the availability of items before allowing an order to be placed, considering stock levels and pending orders.

```
Create or replace trigger validate_stock_before_order.
Before insert on orders
For each row
Begin
  If : new.order_quantity > (select stock_quantity
   from items where item_id = :new.item_id
  End if;
End;
```

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

| 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: /^Wil/i}]})

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: {$elemMatch: {grade: "A", score: 11, date: new Date("2014-08-11")}}},
{restaurants_id: 1, name: 1, grades: 1, _id: 0})

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 {
{ "grade.1": { $ elematch : { grade :"A", score.9, date : new Date ("2014-08-11:00:00
{ restaurants_id : 1, name : 1, grades : 1, _id : 0}.

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}}, {
restaurant_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 restaurant .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, brought : -1})

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

db. restaurants. find ({ "address. street ": { $ exists : false}}).
count() == 0

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]}},
{restaurants_id:1, name:1, grade:1, _id:0});
```

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/i}},
{name:1, borough:1, "address.coord":1, cuisine:1, _id:0});
```

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/i}},
{name:1, borough:1, "address.coord":1, cuisine:1, _id:0});
```

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: {
        $all: [
            { $elemMatch: { score: 2 } },
            { $elemMatch: { score: 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({
    borough: "Manhattan",
    grades: { $all: [{ $elemMatch: {score: 2}}, {$elemMatch:
    {score: 6}}]
```

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({
    borough: ["Manhattan", "brooklyn"]},
    grades: { $all: [{ $elemMatch: {score: 2}}, {$elemMatch:
    {score: 6}}]
})
```

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({
    borough: {$in: ["Manhattan", "Brooklyn"]}
    grades: {$all: [{$elemMatch: {score: 2}}, {$elemMatch:
    {score: 6}}]
}]
```

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.restaurant.find({
grades.{$all:[{$elemMatch:{score:2}},{$elemMatch:{score:6}}]},
borough:{$in:["Manhattan","Brooklyn"]},
cuisine:{$in:["American","Chinese"]}
}).pretty()
```

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:{
        $elemMatch:{
    $or:[{score:2},
        {score: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(
{year:1893},
{_id:0})

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(
    {rated: "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(
    {"imdb.votes": {$gt: 1000}}
)
```

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

```
db.movies.find(
    {"imdb.rating": {$gt: 7.0}}
)
```

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(
    {"award.wins": {$gt: 0}}
)
```

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(
{"award.nominations": {$gt: 0}},
{ title: 1,
  language: 1,
```

released:1,
directors:1,
writers:1,
awards:1,
year:1,
genres:1
}} -id:0

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, language:1, released:1,
directors:1,
writers:1, awards:1, years:1, genres:1, runtime:1, cast:1,
countries:1, _id:
0}}.

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: ISODate("1893-05-09T00:00:00
{title:1, languages:1, released:1, director:1, writers:1,
countries:1, _id:0}}

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},
{title:1, languages:1, released:1,
directors:1, writers:1, countries:1,
_id:0}
)

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

| Ex.No.: 15 | | OTHER DATABASE OBJECTS |
|---|---|---|
| Date: | 19/10/24 | |

## 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

**When to Create an Index**

You should create an index if:
- A column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2 to 4 percent of the rows

**When Not to Create an Index**

It is usually not worth creating an index if:
- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2 to 4 percent of the rows in the table • The table is updated frequently
- The indexed columns are referenced as part of an Expression

**Confirming Indexes**

- The USER_INDEXES data dictionary view contains the name of the index and its uniqueness.
- The USER_IND_COLUMNS view contains the index name, the table name, and the column name.

**EXAMPLE:**

```
SELECT ic.index_name, ic.column_name, ic.column_position col_pos,ix.uniqueness
FROM user_indexes ix, user_ind_columns ic
WHERE ic.index_name = ix.index_name
AND ic.table_name = 'EMPLOYEES';
```

**Removing an Index**

- Remove an index from the data dictionary by using the DROP INDEX command.

- Remove the UPPER_LAST_NAME_IDX index from the data dictionary.

- To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege.

```
DROP INDEX upper_last_name_idx;
```

DROP INDEX *index*;

**Find the Solution for the following:**

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.

2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number

1. Create sequence Dept_id_seq

```
Increment by 10
Start with 200
Max value 1000
Nocache
No cycle;
```

2. Select

```
sequence_name,
max_value;
increment_by
last_number

From
  user_sequence
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, 'Healthcare');
```

4. Create Index emp_dept_id

```
On EMP(Dept_id);
```

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and
Administration. Confirm your additions. Run the commands in your script.
4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table.

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

5. Select

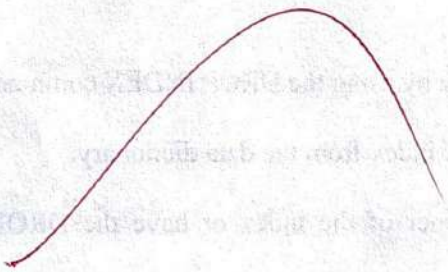
    index_name,
    uniqueness

From

    user_indexes

where

    table_name = 'EMP';

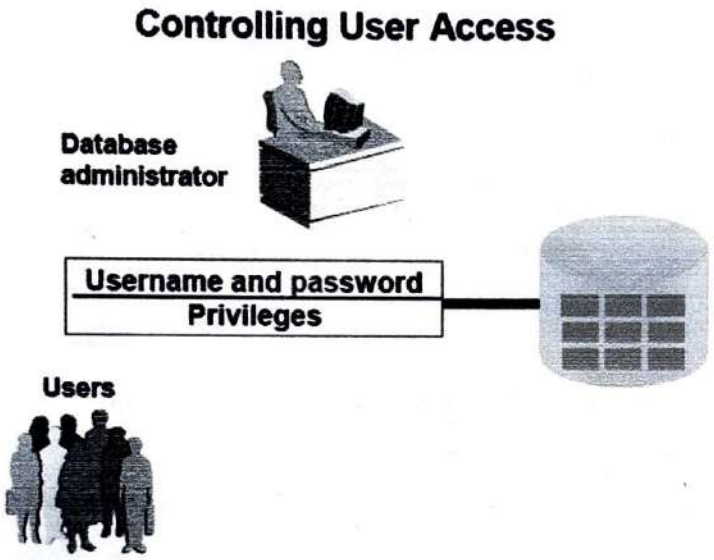


| Ex.No.: 16 | | CONTROLLING USER ACCESS |
|---|---|---|
| Date: | 29/10/24 | |

**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?

_____

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

_____

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

_____

5. What command do you use to change your 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.

1. System Privilage: The create session privilage is classified as a system privilage because it allows the users to establish a connection to the database

2. grant create table to scott;

* grant: This command is used to provide a privilage to a user
* create table: This is the system privilage
* To scott: This specifies the users to whom the privilage is being granted. You can replace scott with any valid username

3. Privileges. Granted by the owner. Granting privileges are the owner can use the with grant option clause to allow the garantee to further pass.

4. Create a role:

create role common_privileges

Grand Privileges:

Grant create session, create table, create view to common_privileges

6. Step-1 : Grant acess to your dept table

2: Grant query access to his or her dept table

Example: commands in sequence

Grant select on dept to John.

8. Step 1: Add new rows

Step 2: Query the other team's table

Select * from dept where dept_ID = 500;

9. Select * from user_tables

10. Revoke the select privilege

Revoke select on depts from team2;

11. * Delete the rows

* COMMIT THE CHANGES;

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