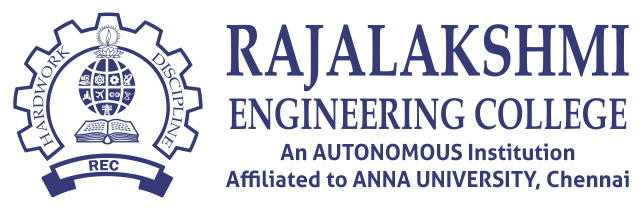
**RAJALAKSHMI ENGINEERING COLLEGE**

**RAJALAKSHMI NAGAR,**

**THANDALAM – 602 105**



CS23332

DATABASE MANAGEMENT SYSTEM

LAB MANUAL

**Laboratory Record Notebook**



# CS23332 DATABASE MANAGEMENT SYSTEMS

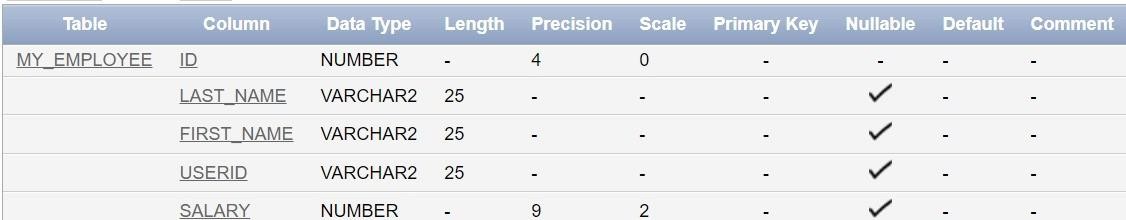
|  |  |
| --- | --- |
| NAME | MONIC AUDITYA.A |
| ROLL NO. | 230701194 |
| DEPT | CSE |
| SEC | C |

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 1** | | **CREATION OF BASE TABLE AND**  **DML OPERATIONS** |
| **Date:** | 31/7/24 |

1. Create MY\_EMPLOYEE table with the following structure

|  |  |  |
| --- | --- | --- |
| 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) NOT NULL, Last\_name VARCHAR2(25), First\_name VARCHAR2(25), Userid VARCHAR2(25),Salary NUMBER(9, 2));



1. 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 |

Begin

INSERT INTO MY\_EMPLOYEE VALUES (1, 'Patel', 'Ralph', 'rpatel', 895); INSERT INTO MY\_EMPLOYEE VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);

End;



1. Display the table with values.

Select \* from My\_Employee;



1. 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.

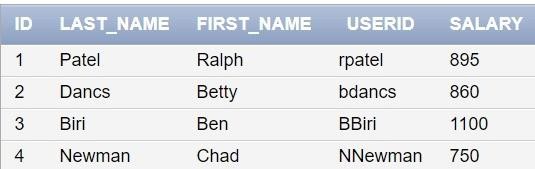
Begin

INSERT INTO MY\_EMPLOYEE (ID, Last\_name, First\_name, Userid, Salary) VALUES (3, 'Biri', 'Ben', SUBSTR('Biri', 1, 1) || SUBSTR('Biri', 1, 7), 1100);

INSERT INTO MY\_EMPLOYEE (ID, Last\_name, First\_name, Userid, Salary)

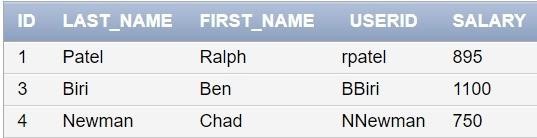
VALUES (4, 'Newman', 'Chad', SUBSTR('Newman', 1, 1) || SUBSTR('Newman', 1, 7), 750);

End;



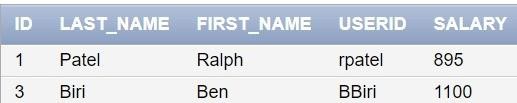
1. Delete Betty dancs from MY \_EMPLOYEE table.

DELETE FROM MY\_EMPLOYEE WHERE Last\_name = 'Dancs';



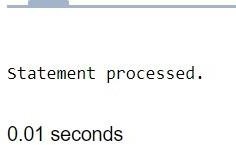
1. Empty the fourth row of the emp table.

DELETE FROM MY\_EMPLOYEE WHERE ID = 4;



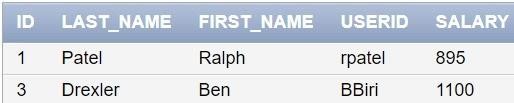
1. Make the data additions permanent.

COMMIT;



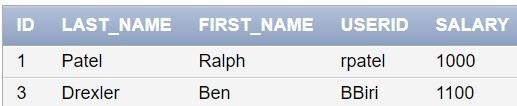
1. Change the last name of employee 3 to Drexler.

UPDATE MY\_EMPLOYEE SET Last\_name = 'Drexler' WHERE ID = 3;



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

UPDATE MY\_EMPLOYEE SET Salary = 1000 WHERE Salary < 900;



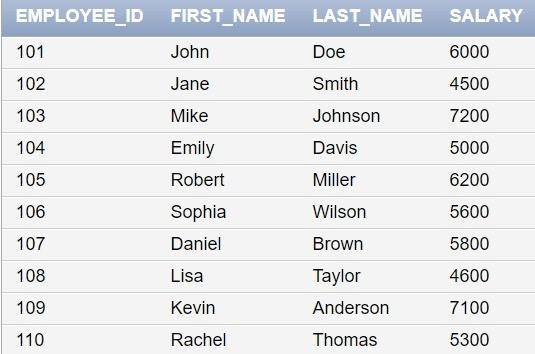
|  |  |  |
| --- | --- | --- |
| **Ex.No.: 2** | | **DATA MANIPULATIONS** |
| **Date:** | 5/8/24 |

### 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) |

1. Find out the employee id, names, salaries of all the employees

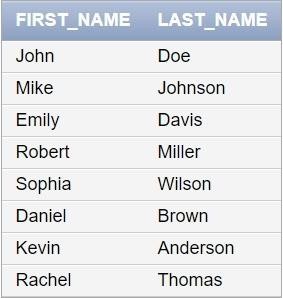
SELECT Employee\_id, First\_name, Last\_name, Salary FROM EMPLOYEES;



1. List out the employees who works under manager 100

SELECT Employee\_id, First\_name, Last\_name FROM EMPLOYEES WHERE Manager\_id = 100;



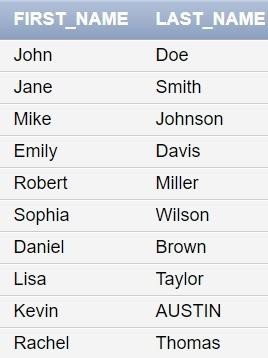
1. Find the names of the employees who have a salary greater than or equal to 4800 SELECT First\_name, Last\_name FROM EMPLOYEES WHERE Salary >= 4800;
2. List out the employees whose last name is ‗AUSTIN‘

SELECT Employee\_id, First\_name, Last\_name FROM EMPLOYEES WHERE Last\_name = 'AUSTIN';



1. 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);



1. 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)

1. Insert Five Records and calculate GrossPay and NetPay.

INSERT INTO EMP (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (1, 'John Doe', 'Manager', 50000, 0.30 \* 50000, -- DA as 30% of Basic

0.40 \* 50000, -- HRA as 40% of Basic,0.12 \* 50000, -- PF as 12% of Basic

50000 + (0.30 \* 50000) + (0.40 \* 50000), -- GrossPay (50000 + (0.30 \* 50000) + (0.40 \*

50000)) - (0.12 \* 50000) -- NetPay

);

INSERT INTO EMP (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (2, 'Jane Smith', 'Clerk', 30000, 0.30 \* 30000, 0.40 \* 30000,

0.12 \* 30000,

30000 + (0.30 \* 30000) + (0.40 \* 30000),

(30000 + (0.30 \* 30000) + (0.40 \* 30000)) - (0.12 \* 30000)

);

INSERT INTO EMP (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (3, 'Mike Johnson', 'Salesman', 40000,

0.30 \* 40000,

0.40 \* 40000,

0.12 \* 40000,

40000 + (0.30 \* 40000) + (0.40 \* 40000),

(40000 + (0.30 \* 40000) + (0.40 \* 40000)) - (0.12 \* 40000)

);

INSERT INTO EMP (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (4, 'Emily Davis', 'Accountant', 35000,

0.30 \* 35000,

0.40 \* 35000,

0.12 \* 35000,

35000 + (0.30 \* 35000) + (0.40 \* 35000),

(35000 + (0.30 \* 35000) + (0.40 \* 35000)) - (0.12 \* 35000)

);

INSERT INTO EMP (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (5, 'Robert Miller', 'Clerk', 25000,

0.30 \* 25000,

0.40 \* 25000,

0.12 \* 25000,

25000 + (0.30 \* 25000) + (0.40 \* 25000),

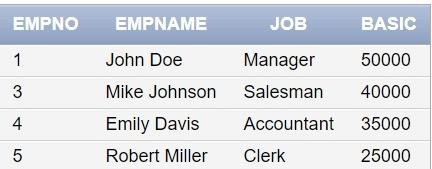
(25000 + (0.30 \* 25000) + (0.40 \* 25000)) - (0.12 \* 25000)

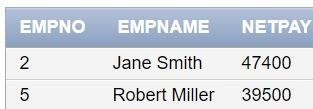
);



1. Display the employees whose Basic is lowest in each department.

SELECT EmpNo, EmpName, Job, Basic FROM EMP E1 WHERE Basic = ( SELECT MIN(Basic) FROM EMP E2 WHERE E2.Job = E1.Job);



1. If Net Pay is less than 50000, display employee number,name and net pay SELECT EmpNo, EmpName, NetPay FROM EMP WHERE NetPay < 50000;

### 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

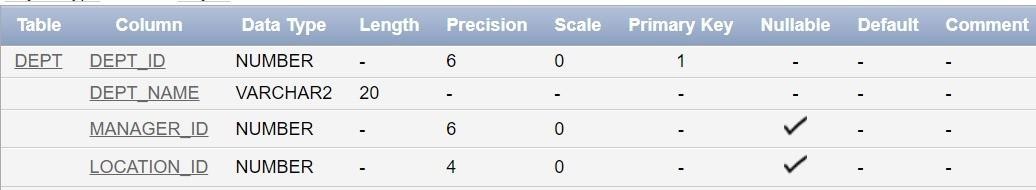
|  |  |  |
| --- | --- | --- |
| **NAME** | **NULL?** | **TYPE** |
| Location\_id | Not null | Number(4) |
| St\_addr |  | Varchar(40) |
| Postal\_code |  | Varchar(12) |
| City | Not null | Varchar(30) |
| State\_province |  | Varchar(25) |
| Country\_id |  | Char(2) |

* 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 (Dept\_id NUMBER(6) NOT NULL, Dept\_name VARCHAR2(20)

NOT NULL,Manager\_id NUMBER(6), Location\_id NUMBER(4), CONSTRAINT my\_dept\_id\_pk PRIMARY KEY (Dept\_id));

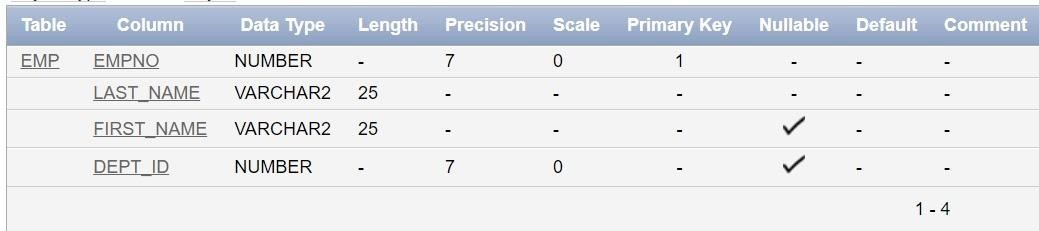


* 1. Create the EMP table based on the following instance chart. Confirm that the table is created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 (EmpNo NUMBER(7) PRIMARY KEY,Last\_name VARCHAR2(25)

NOT NULL,First\_name VARCHAR2(25),Dept\_id NUMBER(7), CONSTRAINT my\_emp\_dept\_id\_fk FOREIGN KEY (Dept\_id) REFERENCES DEPT(Dept\_id));



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

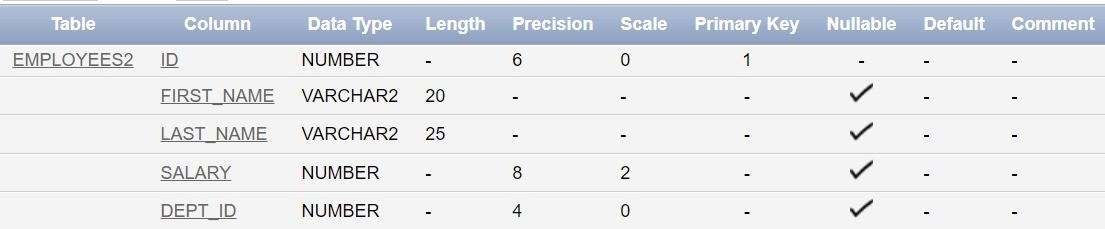
ALTER TABLE EMP MODIFY (Last\_name VARCHAR2(50));



* 1. 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 EMPLOYEES2 (Id NUMBER(6) PRIMARY

KEY,First\_name VARCHAR2(20),Last\_name VARCHAR2(25), Salary NUMBER(8,2),Dept\_id NUMBER(4));



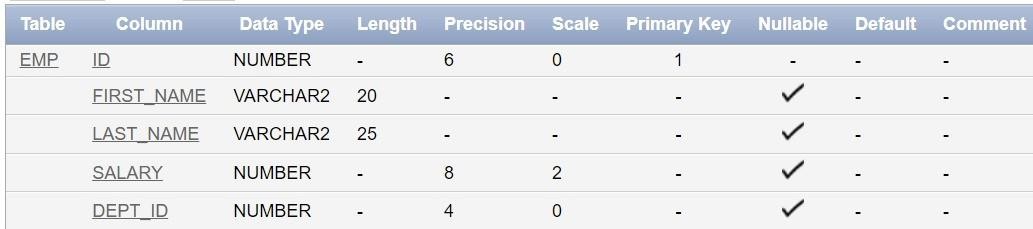
* 1. Drop the EMP Table DROP TABLE EMP;



* 1. Rename the EMPLOYEES2 table as EMP.

ALTER TABLE EMPLOYEES2 RENAME TO EMP;





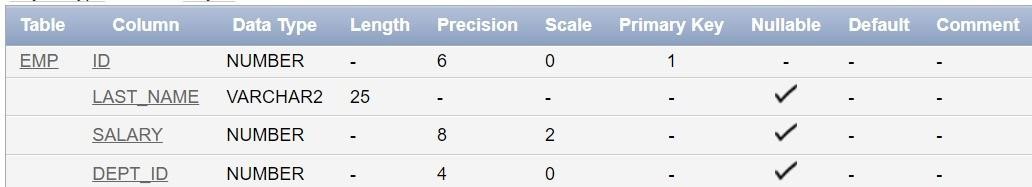
* 1. Add a comment on DEPT and EMP tables. Confirm the modification by describing the table.

COMMENT ON TABLE DEPT IS 'This table contains department information.'; COMMENT ON TABLE EMP IS 'This table contains employee information.';



* 1. Drop the First\_name column from the EMP table and confirm it.

ALTER TABLE EMP DROP COLUMN First\_name;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 3** | | **WRITING BASIC SQL SELECT STATEMENTS** |
| **Date:** | 6/8/24 |

**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;

False ->Corrected Query and Output

Select employee\_id,last\_name,salary\*12 AS "Annual Salary" from Employees;



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

DESC department;

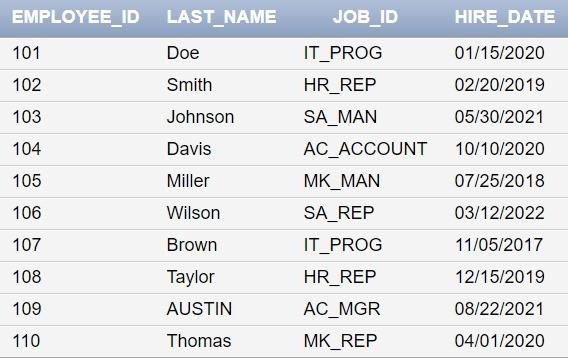


Select \* from Department;



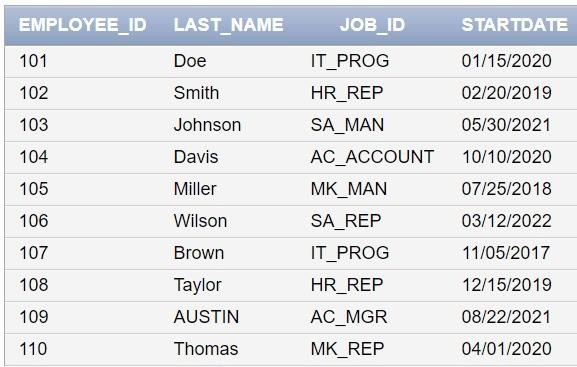
1. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

SELECT employee\_id, last\_name, job\_id, hire\_date FROM employees;



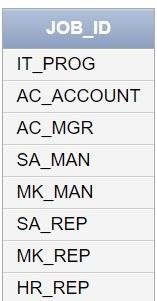
1. Provide an alias STARTDATE for the hire date.

SELECT employee\_id, last\_name, job\_id, hire\_date AS STARTDATE FROM employees;



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

SELECT DISTINCT job\_id FROM employees;



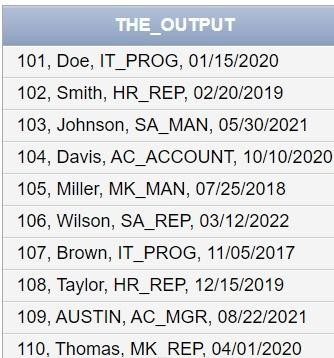
1. Display the last name concatenated with the job ID , separated by a comma and space, and name the column EMPLOYEE and TITLE.

SELECT last\_name || ', ' || job\_id AS "EMPLOYEE and TITLE" FROM employees;



1. Create a query to display all the data from the employees table. Separate each column by a comma. Name the column THE\_OUTPUT.

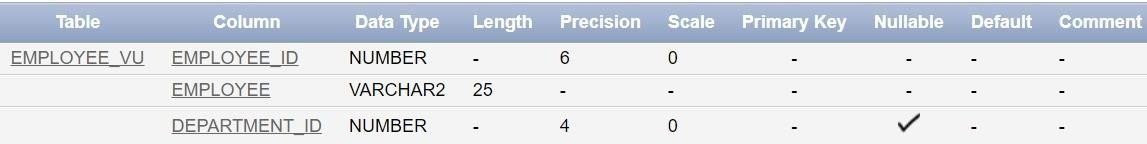
SELECT employee\_id || ', ' || last\_name || ', ' || job\_id || ', ' || hire\_date AS THE\_OUTPUT FROM employees;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 5** | | **CREATING VIEWS** |
| **Date:** | 14/8/24 |

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, last\_name AS EMPLOYEE, department\_id FROM EMPLOYEES;



1. Display the contents of the EMPLOYEES\_VU view.

SELECT \* FROM EMPLOYEE\_VU;



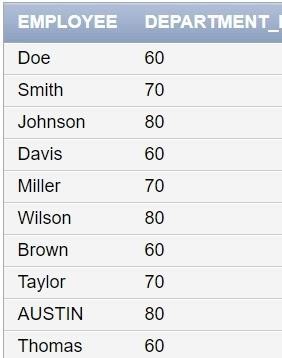
1. Select the view name and text from the USER\_VIEWS data dictionary views.

SELECT view\_name, text FROM USER\_VIEWS WHERE view\_name = 'EMPLOYEE\_VU';



1. Using your EMPLOYEES\_VU view, enter a query to display all employees names and department.

SELECT EMPLOYEE, department\_id FROM EMPLOYEE\_VU;

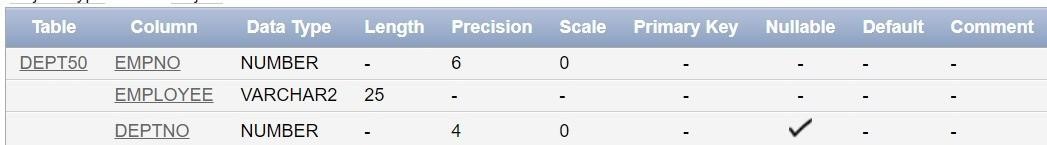


1. Create a view named DEPT50 that contains the employee number, employee last names and department numbers for all employees in department 50.Label the view columns EMPNO, EMPLOYEE and DEPTNO. Do not allow an employee to be reassigned to another department through the view.

CREATE OR REPLACE VIEW DEPT50 (EMPNO, EMPLOYEE, DEPTNO) AS

SELECT employee\_id, last\_name, department\_id FROM EMPLOYEES

WHERE department\_id = 50 WITH CHECK OPTION;



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

SELECT \* FROM DEPT50;



1. Attempt to reassign Matos to department 80.

UPDATE DEPT50 SET DEPTNO = 80 WHERE EMPLOYEE = 'Matos';



1. 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.

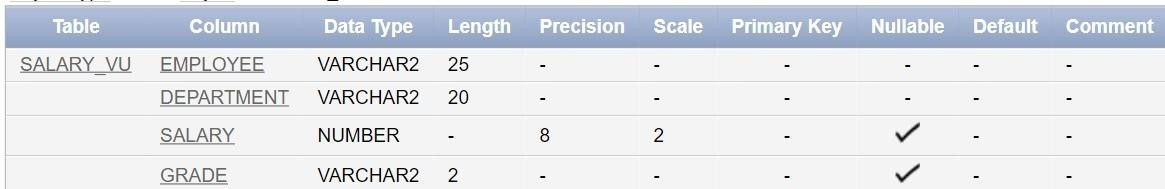
CREATE VIEW SALARY\_VU AS

SELECT e.last\_name AS Employee, d.department\_name AS Department, e.salary AS Salary,

j.grade\_level AS Grade FROM EMPLOYEES e

JOIN DEPARTMENTS d ON e.department\_id = d.department\_id

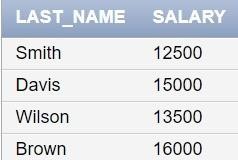
JOIN JOB\_GRADE j ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;



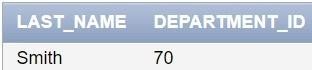
|  |  |  |
| --- | --- | --- |
| **Ex.No.: 6** | | **RESTRICTING AND SORTING DATA** |
| **Date:** | 14/8/24 |

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

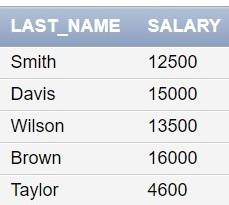
SELECT last\_name, salary FROM employees WHERE salary > 12000;



1. 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;

1. 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 ).

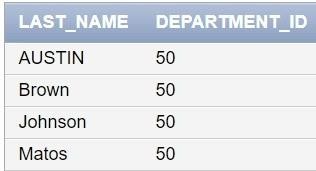
SELECT last\_name, salary FROM employees WHERE salary NOT BETWEEN 5000 AND 12000;

1. 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 '02-20-1998' AND '05-01-1998' ORDER BY hire\_date ASC;

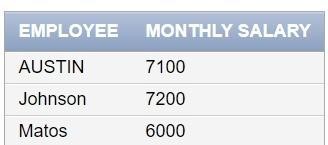


1. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.(hints: in, orderby)

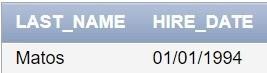
SELECT last\_name, department\_id FROM employees WHERE department\_id IN (20, 50) ORDER BY last\_name ASC;

1. 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)

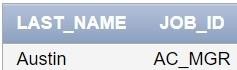
SELECT last\_name AS "EMPLOYEE", salary AS "MONTHLY SALARY" FROM

employees WHERE salary BETWEEN 5000 AND 12000 AND department\_id IN (20, 50) ORDER BY last\_name ASC;

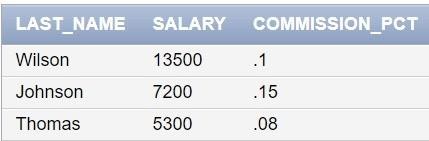
1. Display the last name and hire date of every employee who was hired in 1994.(hints: like) SELECT last\_name, hire\_date FROM employees WHERE hire\_date LIKE '%1994%';



1. 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;

1. 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 employees WHERE commission\_pct IS NOT NULL ORDER BY salary DESC, commission\_pct DESC;

1. 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%';



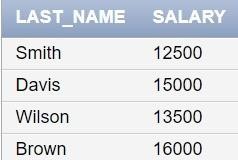
1. Display the last name of all employees who have an a and an ***e*** in their last name.(hints:

like)

SELECT last\_name FROM employees WHERE last\_name LIKE '%a%' AND last\_name LIKE '%e%';

1. 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)

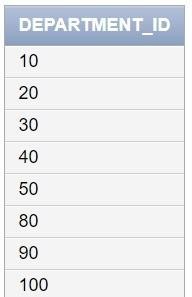
SELECT last\_name, job\_id, salary FROM employees WHERE job\_id IN ('SA\_REP', 'ST\_CLERK') AND salary NOT IN (2500, 3500, 7000);



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 7** | | **USING SET OPERATORS** |
| **Date:** | 28/8/24 |

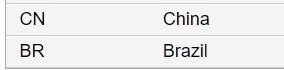
1. The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use set operators to create this report.

SELECT department\_id FROM departments MINUS SELECT department\_id FROM employees WHERE job\_id = 'ST\_CLERK';



1. 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 departments;

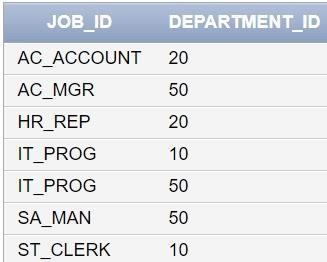


1. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

SELECT job\_id, department\_id FROM employees WHERE department\_id = 10 UNION

SELECT job\_id, department\_id FROM employees WHERE department\_id = 50 UNION

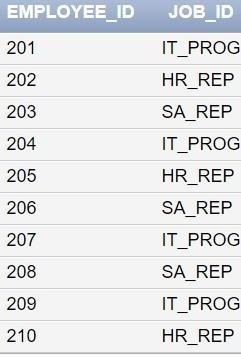
SELECT job\_id, department\_id FROM employees WHERE department\_id = 20;



1. 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 INTERSECT

SELECT employee\_id, job\_id FROM job\_history;

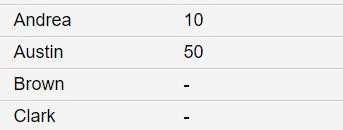


1. The HR department needs a report with the following specifications:

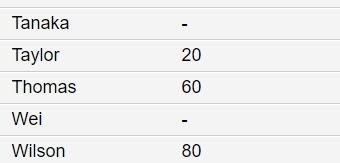
* Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department.
* Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them Write a compound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

SELECT last\_name, department\_id FROM employees UNION

SELECT department\_name, department\_id FROM departments;



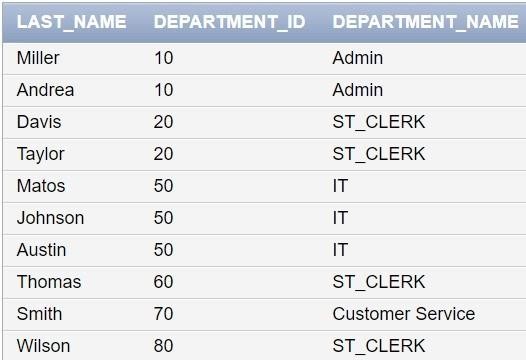




|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ex.No.: 8** | | | **WORKINGWITHMULTIPLETABLES** | |  |
|  | **Date:** | 10/9/24 | |  | |

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

SELECT e.last\_name, e.department\_id, d.department\_name FROM employees e JOIN departments d ON e.department\_id = d.department\_id;



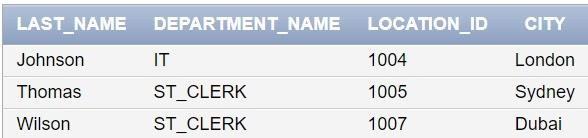
1. Create a unique listing of all jobs that are in department 80. Include the location of the department in the output.

SELECT DISTINCT e.job\_id, d.location\_id FROM employees e JOIN departments d ON e.department\_id = d.department\_id WHERE e.department\_id = 80;



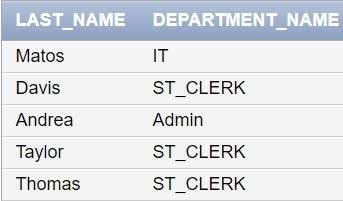
1. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission

SELECT e.last\_name, d.department\_name, d.location\_id, l.city FROM employees e JOIN departments d ON e.department\_id = d.department\_id JOIN locations l ON d.location\_id = l.location\_id WHERE e.commission\_pct IS NOT NULL;



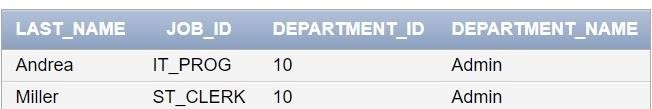
1. Display the employee last name and department name for all employees who have an a(lowercase) in their last names. P

SELECT e.last\_name, d.department\_name FROM employees e JOIN departments d ON e.department\_id = d.department\_id WHERE e.last\_name LIKE '%a%';



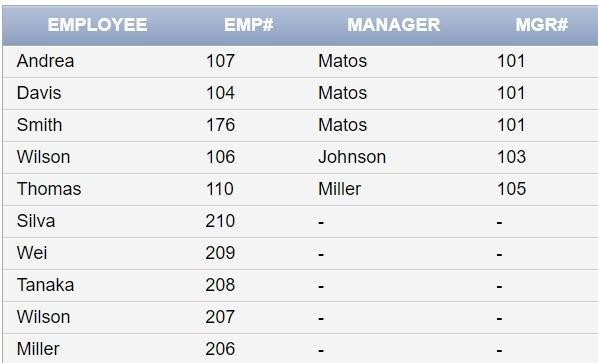
1. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

SELECT e.last\_name, e.job\_id, e.department\_id, d.department\_name FROM employees e JOIN departments d ON e.department\_id = d.department\_id JOIN locations l ON d.location\_id = l.location\_id WHERE l.city = 'Toronto';



1. 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;



1. Modify lab4\_6.sql to display all employees including King, who has no manager. Order the results by the employee number.

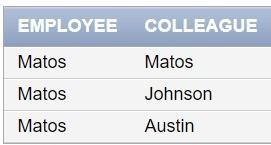
SELECT e.last\_name, e.employee\_id, m.last\_name AS Manager FROM employees e LEFT JOIN employees m ON e.manager\_id = m.employee\_id ORDER BY e.employee\_id;



1. 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

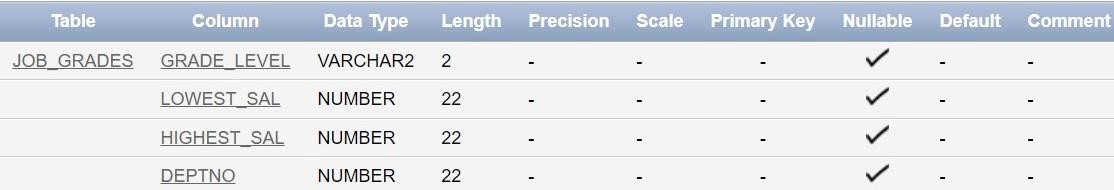
SELECT e1.last\_name AS Employee, e2.last\_name AS Colleague FROM employees e1 JOIN employees e2 ON e1.department\_id = e2.department\_id WHERE e1.employee\_id =

:employee\_id;

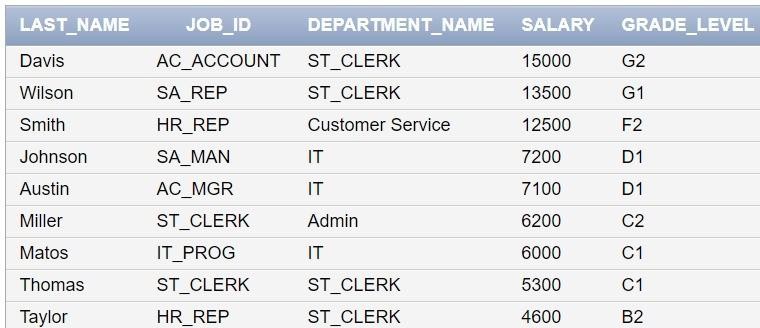


1. Show the structure of the JOB\_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees

DESC job\_grades;



SELECT e.last\_name, e.job\_id, d.department\_name, e.salary, j.grade\_level FROM employees e JOIN departments d ON e.department\_id = d.department\_id JOIN job\_grades j ON e.salary BETWEEN j.lowest\_sal AND j.highest\_sal;



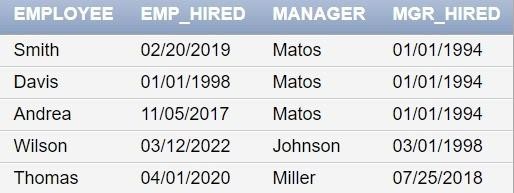
1. Create a query to display the name and hire date of any employee hired after employee Davies.

SELECT last\_name, hire\_date FROM employees WHERE hire\_date > (SELECT hire\_date FROM employees WHERE last\_name = 'Davies');



1. 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.

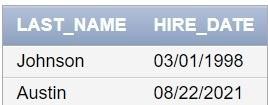
SELECT e.last\_name AS Employee, e.hire\_date AS Emp\_Hired, m.last\_name AS Manager, m.hire\_date AS Mgr\_Hired FROM employees e JOIN employees m ON e.manager\_id = m.employee\_id WHERE e.hire\_date < m.hire\_date;



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ex.No.: 9** | | | **SUB QUERIES** | |  |
|  | **Date:** | 10/9/24 | |  | |

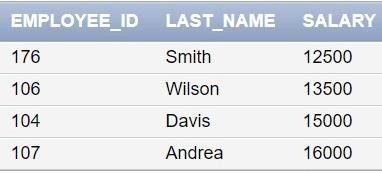
1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters Zlotkey, find all employees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

SELECT e.last\_name, e.hire\_date FROM employees e JOIN employees e2 ON e.department\_id = e2.department\_id WHERE e2.last\_name = :emp\_name AND e.employee\_id != e2.employee\_id;



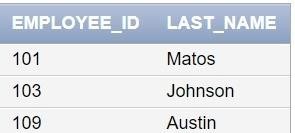
1. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

SELECT employee\_id, last\_name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees) ORDER BY salary ASC;



1. 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*.

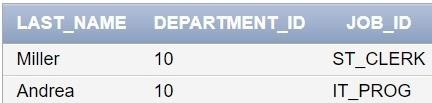
SELECT DISTINCT e1.employee\_id, e1.last\_name FROM employees e1 JOIN employees e2 ON e1.department\_id = e2.department\_id WHERE e2.last\_name LIKE '%u%';



1. 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.department\_id, e.job\_id FROM employees e

JOIN departments d ON e.department\_id = d.department\_id WHERE d.location\_id = 1700;



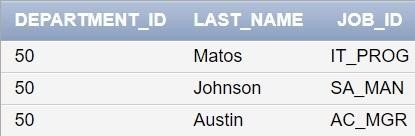
1. Create a report for HR that displays the last name and salary of every employee who reports to King.

SELECT e.last\_name, e.salary FROM employees e JOIN employees m ON e.manager\_id = m.employee\_id WHERE m.last\_name = 'King';



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

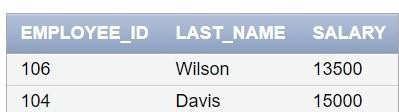
SELECT e.department\_id, e.last\_name, e.job\_id FROM employees e JOIN departments d ON e.department\_id = d.department\_id WHERE d.department\_name = 'Executive';



1. 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 e1.employee\_id, e1.last\_name, e1.salary FROM employees e1

JOIN employees e2 ON e1.department\_id = e2.department\_id WHERE e2.last\_name LIKE '%u%' AND e1.salary > (SELECT AVG(salary) FROM employees);



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 10** | | **AGGREGATING DATA USING GROUP FUNCTIONS** |
| **Date:** | 11/9/24 |

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

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

FALSE

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

TRUE

### The HR department needs the following reports:

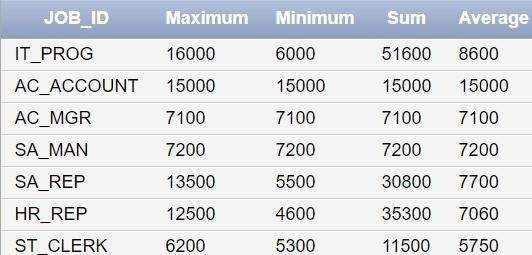
1. Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number

SELECT ROUND(MAX(salary)) AS "Maximum",ROUND(MIN(salary)) AS "Minimum", ROUND(SUM(salary)) AS "Sum", ROUND(AVG(salary)) AS "Average"FROM employees;



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

SELECT job\_id, ROUND(MAX(salary)) AS "Maximum", ROUND(MIN(salary)) AS "Minimum", ROUND(SUM(salary)) AS "Sum", ROUND(AVG(salary)) AS "Average" FROM employees GROUP BY job\_id;



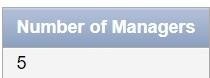
1. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

SELECT COUNT(\*) AS "Number of People" FROM employees WHERE job\_id = '&job\_title';



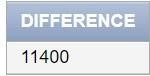
1. Determine the number of managers without listing them. Label the column Number of Managers. *Hint: Use the MANAGER\_ID column to determine the number of managers.*

SELECT COUNT(DISTINCT manager\_id) AS "Number of Managers"FROM employees WHERE manager\_id IS NOT NULL;



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

SELECT (MAX(salary) - MIN(salary)) AS "DIFFERENCE" FROM employees;



1. 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 "Lowest Salary" FROM employees

WHERE manager\_id IS NOT NULL GROUP BY manager\_id HAVING MIN(salary) > 6000 ORDER BY MIN(salary) DESC;



1. 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 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",

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;

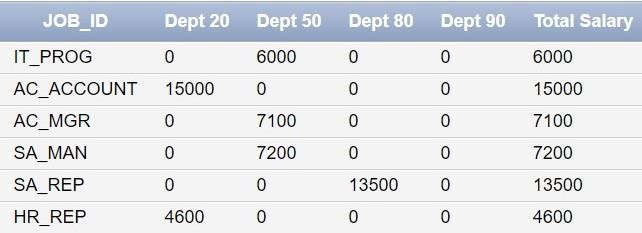


1. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

SELECT job\_id,

SUM(CASE WHEN department\_id = 20 THEN salary ELSE 0 END) AS "Dept 20", SUM(CASE WHEN department\_id = 50 THEN salary ELSE 0 END) AS "Dept 50", SUM(CASE WHEN department\_id = 80 THEN salary ELSE 0 END) AS "Dept 80", SUM(CASE WHEN department\_id = 90 THEN salary ELSE 0 END) AS "Dept 90", SUM(salary) AS "Total Salary"

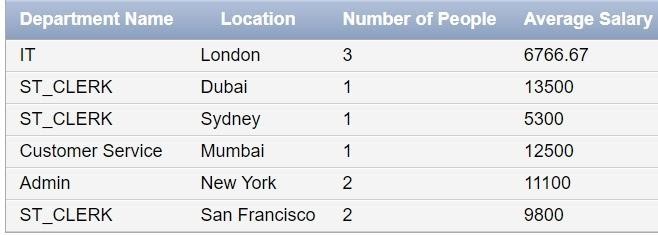
FROM employees WHERE department\_id IN (20, 50, 80, 90) GROUP BY job\_id;



1. 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.department\_name AS "Department Name", l.city AS "Location", COUNT(e.employee\_id) AS "Number of People", ROUND(AVG(e.salary), 2) AS "Average Salary" FROM employees e JOIN departments d ON e.department\_id = d.department\_id JOIN locations l ON d.location\_id = l.location\_id GROUP BY d.department\_name, l.city;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 12** | | **WORKING WITH CURSOR, PROCEDURES AND**  **FUNCTIONS** |
| **Date:** | **23.10.2024** |

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

create or replace function fact (a number) return number is fact number:=1;

b number; begin b:=a; while b>0 loop

fact:=fact\*b; b:=b-1;

end loop; return(fact); end;

/

declare

a number(2); f number(10); begin

a := :n ;

f:=fact(a);

dbms\_output.put\_line('The factorial is '||f); end;

/

Input : 5

Program 2

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

--PROCEDURE FOR IN PARAMETER

create procedure proc(a in number) is bprice number; begin

select price into bprice from library where book\_id=a; dbms\_output.put\_line('The price of the book is '||bprice); end;

declare

a number(2); begin

a := :n;

proc(a); end;

Input: 5

--PROCEDURE FOR OUT PARAMETER

create or replace procedure proc(a in number,n out number) is begin

select publication\_year into n from library where book\_id=a; end;

declare

a number(2); n number(4); begin

a := :b;

proc(a,n);

dbms\_output.put\_line('The year of publication of the book is '||n); end;

Input 7



--PROCEDURE FOR INOUT PARAMETER

create or replace procedure proc(a in out number) is begin

a:=a+10;

end;

declare

a number(2); id number(2); begin

id := :b;

select price into a from library where book\_id=id; proc(a);

dbms\_output.put\_line('The updated price of the book is '||a); end;

Input 3



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

PROGRAM 1

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

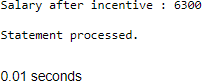
declare

a employees.employee\_id%type; b employees.salary%type;

begin

Select salary into a from employees where employee\_id = 110; b:=0.05\*a;

dbms\_output.put\_line('Salary after incentive : '||(a+b)); end;



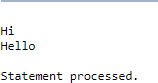
PROGRAM 2

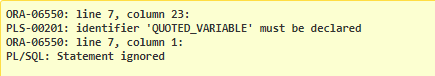
Write a PL/SQL block to show an invalid case-insensitive reference to a quoted and without quoted user-defined identifier.

declare

non\_quoted\_variable varchar2(10) := 'Hi'; "quoted\_variable" varchar2(10) := 'Hello'; begin

dbms\_output.put\_line(NON\_QUOTED\_VARIABLE); dbms\_output.put\_line("quoted\_variable"); dbms\_output.put\_line("QUOTED\_VARIABLE"); end;





PROGRAM 3

Write a PL/SQL block to adjust the salary of the employee whose ID

122. Sample table: employees

declare

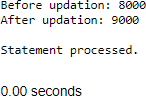
old\_salary employees.salary%type; new\_salary employees.salary%type; begin

new\_salary:= :sal;

Select salary into old\_salary from employees where employee\_id = 122; dbms\_output.put\_line('Before updation: '||old\_salary);

Update employees set salary = salary + new\_salary where employee\_id = 122; Select salary into new\_salary from employees where employee\_id = 122; dbms\_output.put\_line('After updation: '||new\_salary);

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.

Create or replace procedure proc1( a boolean, b boolean) IS BEGIN

if(a is not null) and (b is not null) then if(a = TRUE and b = TRUE) then dbms\_output.put\_line('TRUE');

else dbms\_output.put\_line('FALSE'); end if;

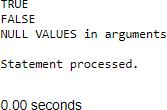
else

dbms\_output.put\_line('NULL VALUES in arguments'); end if;

end proc1;

BEGIN proc1(TRUE,TRUE); proc1(TRUE,FALSE); proc1(NULL,NULL);

end;



PROGRAM 5

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

Declare

name varchar2(20); num number(3);

Begin num := :n;

Select first\_name into name from employees where employee\_id=num; if name like 'D%' then

dbms\_output.put\_line('Name starts with "D"'); end if;

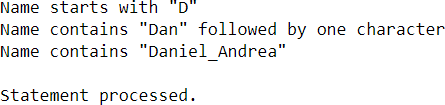
if name like 'Dan\_el%' then

dbms\_output.put\_line('Name contains "Dan" followed by one character'); end if;

name := 'Daniel\_Andrea';

if name like 'Daniel\\_Andrea' escape '\' then dbms\_output.put\_line('Name contains "Daniel\_Andrea"'); end if;

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

a number(2); b number(2);

num\_small number(2); num\_large number(2); begin

a := :s;

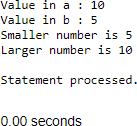
b := :l;

dbms\_output.put\_line('Value in a : '||a); dbms\_output.put\_line('Value in b : '||b); if a>b then

num\_small := b; num\_large := a; else

num\_small :=a; num\_large :=b; end if;

dbms\_output.put\_line('Smaller number is '||num\_small); dbms\_output.put\_line('Larger number is '||num\_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.

Create or replace procedure calc\_incen(emp\_id number,achievement number,target number) AS

incentive number; rowcount number;

Begin

if achievement > target then incentive:= achievement\*0.2; else

incentive:=0; end if;

Update employees set salary = salary + incentive where employee\_id = emp\_id; rowcount:= SQL%ROWCOUNT;

if rowcount>0 then dbms\_output.put\_line('Record(s) updated'); else

dbms\_output.put\_line('No Record(s) updated'); end if;

end;

Declare

id number; achievement number; target number;

Begin

id := :emp\_id; achievement := :achieve; target := :target\_;

calc\_incen(id,achievement,target); end;



PROGRAM 8

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

Create or replace procedure calc\_incen(emp\_id number,sales number) AS incentive number;

rowcount number;

Begin

if sales < 1000 then incentive:= 0;

elsif sales > 1000 and sales < 2000 then incentive := sales \* 0.2;

else

incentive := sales \* 0.5; end if;

Update employees set salary = salary + incentive where employee\_id = emp\_id; rowcount:= SQL%ROWCOUNT;

if rowcount>0 then dbms\_output.put\_line('Record(s) updated'); else

dbms\_output.put\_line('No Record(s) updated'); end if;

end;

Declare

id number; sales number; sal number;

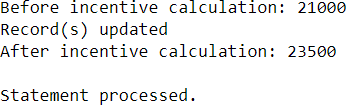
Begin

id := :emp\_id; sales := :sale;

select salary into sal from employees where employee\_id = id; dbms\_output.put\_line('Before incentive calculation: '||sal); calc\_incen(id,sales);

select salary into sal from employees where employee\_id = id; dbms\_output.put\_line('After incentive calculation: '||sal);

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

emp\_count number; vacancy number := 20; begin

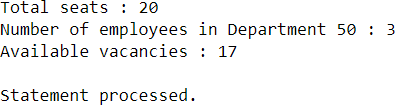
Select count(\*) into emp\_count from employees where department\_id = 10; dbms\_output.put\_line('Total seats : '||vacancy);

dbms\_output.put\_line('Number of employees in Department 50 : '||emp\_count); if emp\_count>vacancy then

dbms\_output.put\_line('No vacancies available'); else

dbms\_output.put\_line('Available vacancies : '||(vacancy-emp\_count)); 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

dept\_id number; emp\_count number; vacancy number := 10; begin

dept\_id := :id;

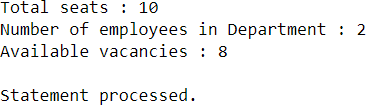
Select count(\*) into emp\_count from employees where department\_id = dept\_id; dbms\_output.put\_line('Total seats : '||vacancy);

dbms\_output.put\_line('Number of employees in Department : '||emp\_count); if emp\_count>vacancy then

dbms\_output.put\_line('No vacancies available'); else

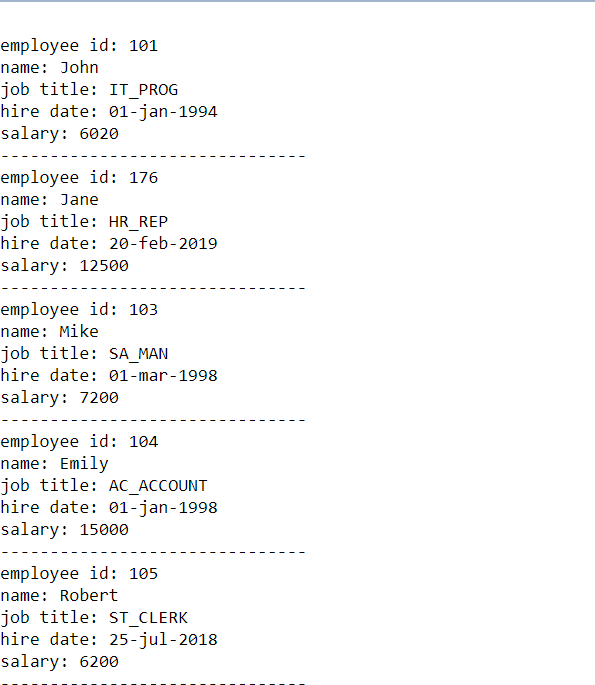
dbms\_output.put\_line('Available vacancies : '||(vacancy-emp\_count)); end if;

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 i in (select employee\_id, first\_name, job\_id, hire\_date, salary from employees) loop

dbms\_output.put\_line('employee id: ' || i.employee\_id); dbms\_output.put\_line('name: ' || i.first\_name); dbms\_output.put\_line('job title: ' || i.job\_id);

dbms\_output.put\_line('hire date: ' || to\_char(i.hire\_date, 'dd-mon-yyyy')); dbms\_output.put\_line('salary: ' || i.salary);

dbms\_output.put\_line(' ');

end loop; end;

PROGRAM 12

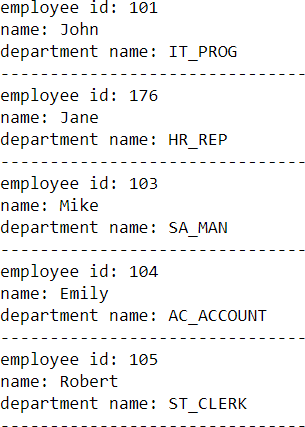
Write a PL/SQL program to display the employee IDs, names, and department names of all employees.

begin

for i in (select e.employee\_id, e.first\_name, e.job\_id from employees e) loop

dbms\_output.put\_line('employee id: ' || i.employee\_id); dbms\_output.put\_line('name: ' || i.first\_name); dbms\_output.put\_line('department name: ' || i.job\_id); dbms\_output.put\_line(' ');

end loop; end;



PROGRAM 13

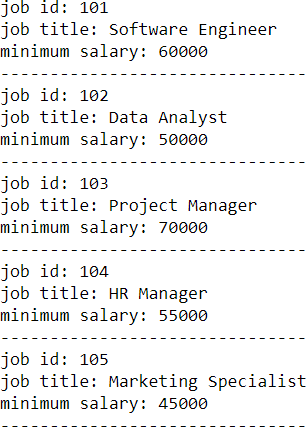
Write a PL/SQL program to display the job IDs, titles, and minimum salaries of all jobs.

Begin

for i in (select job\_id,job\_title,min\_salary from jobs) loop

dbms\_output.put\_line('job id: ' || i.job\_id); dbms\_output.put\_line('job title: ' || i.job\_title); dbms\_output.put\_line('minimum salary: ' || i.min\_salary); dbms\_output.put\_line(' ');

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 i in (select employee\_id,employee\_name,start\_date from job\_history) loop

dbms\_output.put\_line('employee id: ' || i.employee\_id); dbms\_output.put\_line('name: ' || i.employee\_name); dbms\_output.put\_line('start date: ' ||to\_char(i.start\_date, 'dd-mon-yyyy')); dbms\_output.put\_line(' ');

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 i in (select employee\_id,employee\_name,end\_date from job\_history) loop

dbms\_output.put\_line('employee id: ' || i.employee\_id); dbms\_output.put\_line('name: ' || i.employee\_name); dbms\_output.put\_line('end date: ' ||to\_char(i.end\_date, 'dd-mon-yyyy')); dbms\_output.put\_line(' ');

end loop; end;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 13** | | **WORKING WITH TRIGGER TRIGGER** |
| **Date:** | **29.10.2024** |

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

child\_count NUMBER; BEGIN

SELECT COUNT(\*) INTO child\_count FROM child\_table

WHERE parent\_id = :OLD.parent\_id;

IF child\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete parent record as child records exist.');

END IF;

END;

Testing of Trigger

DELETE FROM parent\_table WHERE parent\_id = 1;

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\_duplicate\_value BEFORE INSERT OR UPDATE ON table\_name

FOR EACH ROW DECLARE

v\_count NUMBER; BEGIN

-- Check if the new value already exists in the table SELECT COUNT(\*) INTO v\_count

FROM table\_name

WHERE specific\_column = :NEW.specific\_column;

-- If a duplicate is found, raise an error IF v\_count > 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Duplicate value detected in specific column.'); END IF;

END;

/

**Output:**

ORA-20002: Duplicate value detected in specific column.

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\_insertion BEFORE INSERT ON table\_name

FOR EACH ROW DECLARE

v\_total NUMBER;

v\_threshold CONSTANT NUMBER := 10000; -- Set your threshold here BEGIN

-- Calculate the total sum of the column values

SELECT SUM(column\_name) INTO v\_total FROM table\_name;

-- Prevent insertion if the threshold is exceeded

IF v\_total + :NEW.column\_name > v\_threshold THEN RAISE\_APPLICATION\_ERROR(-20003, 'Cannot insert, total column value

exceeds threshold.');

END IF;

END;

/

## Output:

ORA-20003: Cannot insert, total column value exceeds threshold.

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\_column\_changes AFTER UPDATE ON table\_name

FOR EACH ROW BEGIN

-- Check if specific columns have been modified

IF :OLD.column\_name1 != :NEW.column\_name1 OR :OLD.column\_name2 !=

:NEW.column\_name2 THEN

-- Insert the old and new values into the audit table

INSERT INTO audit\_table (user\_id, change\_time, old\_value, new\_value) VALUES (USER, SYSDATE, :OLD.column\_name1 || ', ' || :OLD.column\_name2,

:NEW.column\_name1 || ', ' || :NEW.column\_name2); END IF;

END;

/

Output:

### User\_ID Change\_Time Old\_Value New\_Value

SYSTEM 2024-09-19

10:05:00

OldValue1, OldValue2

NewValue, AnotherNewValue

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\_user\_activity AFTER INSERT OR UPDATE OR DELETE ON table\_name FOR EACH ROW

BEGIN

IF INSERTING THEN

INSERT INTO audit\_log (user\_id, operation, record\_id, change\_time) VALUES (USER, 'INSERT', :NEW.id\_column, SYSDATE);

ELSIF UPDATING THEN

INSERT INTO audit\_log (user\_id, operation, record\_id, change\_time) VALUES (USER, 'UPDATE', :NEW.id\_column, SYSDATE);

ELSIF DELETING THEN

INSERT INTO audit\_log (user\_id, operation, record\_id, change\_time) VALUES (USER, 'DELETE', :OLD.id\_column, SYSDATE);

END IF;

END;

/

|  |  |  |  |
| --- | --- | --- | --- |
| Output:  **User\_ID** | **Operation** | **Record\_ID** | **Change\_Time** |
| SYSTEM | INSERT | 1 | 2024-09-19  10:10:00 |
| SYSTEM | UPDATE | 1 | 2024-09-19  10:15:00 |
| SYSTEM | DELETE | 1 | 2024-09-19  10:20:00 |

Program 6

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\_running\_total AFTER INSERT ON table\_name

FOR EACH ROW BEGIN

-- Update the running total column in the total\_table UPDATE total\_table

SET running\_total = running\_total + :NEW.value\_column WHERE total\_id = :NEW.total\_id;

END;

/

Output:

### Total\_ID Running\_Total

1 1500

Program 7

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\_item\_availability BEFORE INSERT ON orders

FOR EACH ROW DECLARE

v\_stock\_level NUMBER; v\_pending\_orders NUMBER; BEGIN

SELECT stock INTO v\_stock\_level FROM inventory WHERE item\_id = :NEW.item\_id;

-- Check pending orders

SELECT SUM(quantity) INTO v\_pending\_orders FROM orders

WHERE item\_id = :NEW.item\_id AND status = 'Pending';

-- Ensure stock is available for the order

IF v\_stock\_level - v\_pending\_orders < :NEW.order\_quantity THEN RAISE\_APPLICATION\_ERROR(-20004, 'Insufficient stock available for this

order.'); END IF; END;

/

### Output:

ORA-20004: Insufficient stock available for this order.

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 14** | | **MongoDB** |
| **Date:** | 30/10/24 |

**Restaurant Collection**

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

db.restaurants.find(

{

$or: [

{

'Chinese'

},

{

cuisine: { $nin: ['American', 'Chinese'] } // Cuisines other than 'American' and

name: { $regex: '^Wil', $options: 'i' } // Restaurant names that begin with 'Wil'

(case-insensitive)

}

]

},

{

\_id: 1, // Retrieve the restaurant ID name: 1, // Retrieve the restaurant name borough: 1, // Retrieve the borough cuisine: 1 // Retrieve the cuisine

}

)

1. **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 "2014-08-11 T00:00:00Z" among many of survey dates.**

db.restaurants.find(

{

"grades": {

$elemMatch: {

"grade": "A", // Grade must be "A"

"score": 11, // Score must be 11

"date": ISODate("2014-08-11T00:00:00Z") // Date must match the specified

ISODate

}

}

},

{

\_id: 1, // Retrieve the restaurant ID name: 1, // Retrieve the restaurant name grades: 1 // Retrieve the grades

}

)

1. **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": { // Accessing the 2nd element (index 1) of the grades array "grade": "A", // Grade must be "A"

"score": 9, // Score must be 9

"date": ISODate("2014-08-11T00:00:00Z") // Date must match the specified ISODate

}

},

{

\_id: 1, // Retrieve the restaurant ID name: 1, // Retrieve the restaurant name grades: 1 // Retrieve the grades

}

)

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

db.restaurants.find(

{

"coord.1": { $gt: 42, $lte: 52 } // Accessing the 2nd element (index 1) of the coord array

},

{

\_id: 1, // Retrieve the restaurant ID name: 1, // Retrieve the restaurant name address: 1, // Retrieve the address

coord: 1 // Retrieve the geographical location (coord)

}

)

1. **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 })

1. **Write a mongoDB query to arrange the name of the restaurants in descending order along with all the columns.**

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

1. **Write a MongoDB query to arrange 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 })

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

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

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

db.restaurants.find({

"coord": { $type: "double" } // or you can use $type: 1

})

1. **Write a mongoDB query which will select the restaurant Id, name and grades for those restaurants which return 0 as a remainder after dividing the score by 7.**

db.restaurants.find(

{

"grades": {

$elemMatch: {

$expr: {

$eq: [{ $mod: ["$score", 7] }, 0] // Check if score % 7 == 0

}

}

}

},

{

\_id: 1, // Retrieve the restaurant ID name: 1, // Retrieve the restaurant name grades: 1 // Retrieve the grades

}

)

1. **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 } // Regex to find 'mon' anywhere in the name (case-insensitive)

},

{

name: 1, // Retrieve the restaurant name borough: 1, // Retrieve the borough

"coord.0": 1, // Retrieve longitude (assuming longitude is the first element in the coord array)

"coord.1": 1, // Retrieve latitude (assuming latitude is the second element in the coord array)

cuisine: 1, // Retrieve the cuisine

\_id: 0 // Exclude the restaurant ID from the results

}

)

1. **Write a mongodb query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contains 'Mad' as first three letters in its name.**

db.restaurants.find(

{

name: { $regex: /^Mad/i } // Regex to find names starting with 'Mad' (case-insensitive)

},

{

name: 1, // Retrieve the restaurant name borough: 1, // Retrieve the borough

"coord.0": 1, // Retrieve longitude (assuming longitude is the first element in the coord array)

"coord.1": 1, // Retrieve latitude (assuming latitude is the second element in the coord array)

cuisine: 1, // Retrieve the cuisine

\_id: 0 // Exclude the restaurant ID from the results

}

)

1. **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": {

$elemMatch: {

score: { $lt: 5 } // Score must be less than 5

}

}

}

)

1. **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(

{

borough: "Manhattan", // Condition to filter by borough "grades": {

$elemMatch: {

score: { $lt: 5 } // Condition to filter grades with score less than 5

}

}

}

)

1. **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(

{

$or: [

{ borough: "Manhattan" }, // Condition to filter by borough Manhattan

{ borough: "Brooklyn" } // Condition to filter by borough Brooklyn

],

"grades": {

$elemMatch: {

score: { $lt: 5 } // Condition to filter grades with score less than 5

}

}

}

)

}

)

# Movies Collection

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

db.movies.find(

{

releaseYear: 1893 // Assuming the field for the release year is named 'releaseYear'

}

)

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

db.movies.find(

{

runtime: { $gt: 120 } // Assuming the field for runtime is named 'runtime'

}

)

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

db.movies.find(

{

genres: "Short" // Assuming the field for genres is an array named 'genres'

}

)

1. **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(

{

director: "William K. L. Dickson" // Assuming the field for the director is named 'director'

}

)

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

db.movies.find(

{

country: "USA" // Assuming the field for the release country is named 'country'

}

)

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

db.movies.find(

{

rating: "UNRATED" // Assuming the field for the rating is named 'rating'

}

)

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

db.movies.find(

{

votes: { $gt: 1000 } // Assuming the field for votes is named 'votes'

}

)

1. **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 } // Assuming the field for IMDb rating is named 'imdbRating'

}

)

1. **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: { $gt: 4 } } // Assuming the viewer rating is nested within a 'tomatoes' object

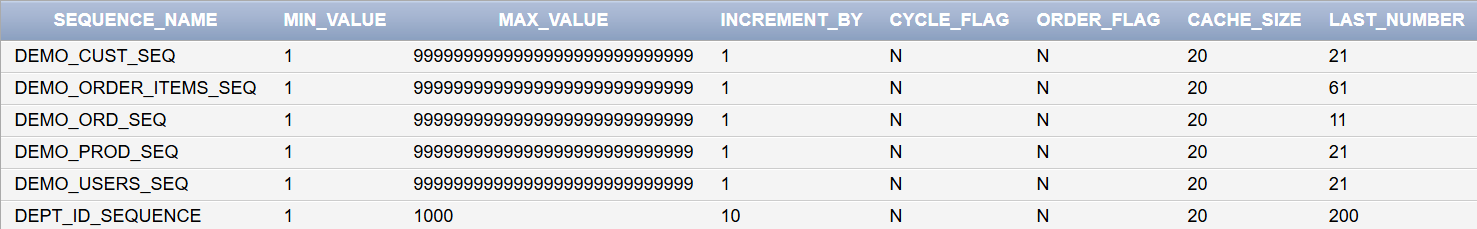
}

)

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 15** | | **OTHER DATABASE OBJECTS** |
| **Date:** | **04.11.2024** |

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.

Create Sequence dept\_id\_sequence start with 200 increment by 10 maxvalue 1000;



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

SELECT sequence\_name, max\_value,increment\_by AS increment\_size,last\_number FROM user\_sequences WHERE sequence\_name = 'DEPT\_ID\_SEQUENCE';

1. 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.

Insert into departments values(dept\_id\_sequence.nextval,'HR',111,1010,'US','United States'); Insert into departments values(dept\_id\_seq.nextval,'Admin',112,1011,'IN','India');



1. Create a nonunique index on the foreign key column (DEPT\_ID) in the EMP table.

Create index emp\_dept\_index on Employees(department\_id);



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

SELECT index\_name, uniqueness FROM user\_indexes WHERE table\_name = 'Employees'; Output :

Index\_name : EMPLOYEE\_INDEX Uniqueness : NONUNIQUE

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 16** | | **CONTROLLING USER ACCESS** |
| **Date:** | **06.11.2024** |

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

**~** The privilege is CREATE SESSION. This privilege allows a user to log on to the Oracle Server. It is a system privilege, not an object privilege.

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

**~** The privilege is CREATE TABLE . This is a system privilege that allows a user to create tables in their own schema.

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

**~** The owner of the table (the user who created it) can pass along privileges to other users. This is done using the GRANT command.

For example:

GRANT SELECT ON my\_table TO other\_user;

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

**~** Use a role to bundle common system privileges.

Assign this role to users instead of granting privileges individually.

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

**~** ALTER USER username IDENTIFIED BY new\_password;

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

**~** GRANT SELECT ON DEPARTMENTS TO other\_user;

**~** GRANT SELECT ON DEPARTMENTS TO your\_username;

1. Query all the rows in your DEPARTMENTS table.

**~** SELECT \* FROM DEPARTMENTS;

1. 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.

**~** Team 1 should execute:

INSERT INTO DEPARTMENTS (DEPARTMENT\_ID, DEPARTMENT\_NAME) VALUES

(500, 'Education'); COMMIT;

**~** Team 2 should execute:

INSERT INTO DEPARTMENTS (DEPARTMENT\_ID, DEPARTMENT\_NAME) VALUES

(510, 'Human Resources');

COMMIT;

**~** To query the other team’s table:

SELECT \* FROM other\_user.DEPARTMENTS;

1. Query the USER\_TABLES data dictionary to see information about the tables that you own.

**~** SELECT \* FROM USER\_TABLES;

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

**~** To revoke the SELECT privilege on your table from the other team: REVOKE SELECT ON DEPARTMENTS FROM other\_user;

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

**~** To remove the row you inserted into the DEPARTMENTS table and save the changes: For Team 1 (removing the Education department with ID 500):

DELETE FROM DEPARTMENTS WHERE DEPARTMENT\_ID = 500; COMMIT;

For Team 2 (removing the Human Resources department with ID 510): DELETE FROM DEPARTMENTS WHERE DEPARTMENT\_ID = 510; COMMIT;