**RAJALAKSHMI ENGINEERING**

**COLLEGE RAJALAKSHMI NAGAR,**

**THANDALAM – 602 105**



**CS23332 DATABASE MANAGEMENT**

**SYSTEMS LAB**

**Laboratory Record Notebook**

Name: RAMESH M

Year / Branch / Section: II/AIDS/C

University Register No: 2116231801134

College Roll No: 231801134

Semester: III

Academic Year: 2024-2025

# CS23332 DATABASE MANAGEMENT SYSTEMS

|  |  |
| --- | --- |
| **NAME** | **RAMESH M** |
| **ROLL NO.** | **2116231801134** |
| **DEPT** | **AI&DS** |
| **SEC** | **‘C’** |

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 1** | | **CREATION OF BASE TABLE AND DML OPERATIONS** |
| **Date:** | 01/08/2024 |

1. Create MY\_EMPLOYEE table with the following structure

CREATE TABLE MY\_EMPLOYEE( ID Number(4) NOT NULL,

Last\_name Varchar(25),

First\_name Varchar(25),

Userid Varchar(25),

Salary Number(9,2)

);



1. Add the first row and second rows data to MY\_EMPLOYEE table from the sample table

Insert



MY\_EMPLOYEE(&ID,&LAST\_NAME,&FIRST\_NAME,&USERID,&SALARY

)

values(1,”Patel”,”Ralph”,”rpatel”,895 2,”Dancs”,”Betty”,”bdancs”,860);

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 first seven letters of the last\_name to produce Userid

Update MY\_EMPLOYEES

Set Userid = substr(first\_name,1,1) || substr(last\_name,1,7)

Where ID in (3,4);

1. delete Betty dancs from my\_employee

table`1 Delete from MY\_EMPLOYEE

Where FIRST\_NAME = ‘Betty’ and LAST\_NAME = ‘Dancs’;



1. Empty the fourth row of the emp table

Delete from MY\_EMPLOYEE

Where ID = 5;



1. Make the data additions permanent Commit;
2. 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:** | 08/08/2024 |

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

Employee\_id, First\_Name, Salary from EMPLOYEES;



1. List out the employees who works under manager 100

select First\_Name || ' ' || Last\_Name as 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 as name from EMPLOYEES Where salary >= 4800;



1. List out the employees whose last name is ̳AUSTIN

select First\_Name || ' ' || Last\_Name as 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 as name from EMPLOYEES

where Department\_id in (60,70,80);



1. Display the unique Manager\_Id.

select DISTINCT(manager\_id) from EMPLOYEES;



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

INSERT INTO Emp (EmpNo, EmpName, Job, Basic, DA, HRA, PF, GrossPay, NetPay) VALUES (

101, 'John Doe', 'Manager', 50000, 15000, 20000, 6000,0,0 ,

102, 'Jane Smith', 'Developer', 40000, 12000, 16000, 4800,0,0 ,

103, 'Alice Johnson', 'Analyst', 35000, 10500, 14000, 4200,0,0 ,

104, 'Bob Brown', 'Designer', 30000, 9000, 12000, 3600,0,0 ,

105, 'Charlie Davis', 'Tester', 25000, 7500, 10000, 3000,0,0

)

update emp

set GrossPay = Basic+DA+HRA where Grosspay = 0;

update emp

set NetPay = Grosspay - PF where Netpay = 0;

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

select job,min(basic) from Emp group by Job;



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

Create table DEPT(

ID Number(7),

Name varchar(25)

);

Desc DEPT;



1. Create the EMP1 table based on the following instance chart. Confirm that the table is created.

create table EMP1(

ID Number(7),

First\_name varchar(25),

Last\_name varchar(25),

Dept\_id Number(7)

);

Desc EMP1;



1. Modify the EMP1 table to allow for longer employee last names. Confirm the modification.(Hint: Increase the size to 50)

ALTER TABLE EMP1

modify Last\_name varchar(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(10),

First\_name varchar(50),

Last\_name varchar(50),

Salary Number(10),

Dept\_id Number(10)

);

1. Drop the EMP1 table.

drop table EMP1; 6) Rename the EMPLOYEES2 table as EMP1.

ALTER TABLE EMPLOYEES2 RENAME TO EMP1;

1. Add a comment on DEPT and EMP1 tables. Confirm the modification by describing the table.

comment on TABLE DEPT IS 'this table contains the fields ID and NAME..';

SELECT TABLE\_NAME, COMMENTS

FROM USER\_TAB\_COMMENTS

WHERE TABLE\_NAME = 'DEPT';



comment on TABLE EMP1 IS 'this table contains the fields ID,first name,last name,salary,DEPT\_id..';

SELECT TABLE\_NAME, COMMENTS

FROM USER\_TAB\_COMMENTS

WHERE TABLE\_NAME = 'EMP1';



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

ALTER TABLE EMP1

drop column First\_name;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 3** | | **WRITING BASIC SQL SELECT STATEMENTS** |
|  |  |
| **Date:** | 10/08/2024 |  |

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

The columns in select statement should be separated by commas and the column alias should be given by using a keyword “as”

SELECT employee\_id, last\_name, salary\*12 as "ANNUAL SALARY" FROM employees;



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

Desc employees;



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

select employee\_id , job\_id , last\_name , hire\_date from employees;



1. Provide an alias STARTDATE for the hire date. select 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;



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

select employee\_id || ' , ' || first\_name || ' , ' || last\_name || ' , ' || email || ' , ' || phone\_number || ' ,

' || hire\_date || ' , ' || job\_id || ' , ' || salary || ' , ' || commission\_pct || ' , ' || manager\_id || ' , ' || department\_id as "THE\_OUTPUT" from employees;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 4** | | **WORKING WITH CONSTRAINTS** |
| **Date:** | 16/08/2024 |

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 EMP1 add constraint my\_emp\_id\_pk PRIMARY KEY(ID);

1. Create a PRIMAY KEY constraint to the DEPT table using the ID colum. The constraint should be named at creation. Name the constraint my\_dept\_id\_pk.

alter table DEPT

add constraint my\_dept\_id\_pk PRIMARY KEY(ID);

1. 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 Numbe(10);

alter table emp

add constraint my\_emp\_dept\_id\_fk FOREIGN KEY(DEPT\_ID) references dept(ID);

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

alter table emp add COMMISSION Number(2,2);

alter table emp

add CONSTRAINT commission\_gt\_zero CHECK(COMMISSION > 0);

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 5** | | **CREATING VIEWS** |
|  |  |
| **Date:** | 23/08/2024 |  |

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

create view EMPLOYEE\_VU as

select employee\_id , first\_name || ' ' || last\_name as "EMPLOYEE", department\_id from 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 VIEW DEPT50 AS

SELECT employee\_id AS EMPNO, employee AS EMPLOYEE,

department\_id AS DEPTNO

FROM EMPLOYEE\_VU

WHERE department\_id = 50

WITH READ ONLY;



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

Desc dept50;



1. Attempt to reassign Matos to department 80.

UPDATE EMPLOYEES

SET department\_id = 80

WHERE first\_name = '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.dept\_name AS Department,

e.salary AS Salary,

j.grade\_level AS Grade

FROM EMPLOYEES e

JOIN DEPARTMENT d

ON e.department\_id = d.dept\_id

JOIN JOB\_GRADE j

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



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 6** | | **RESTRICTING AND SORTING DATA** |
|  |  |
| **Date:** | 29/08/2024 |  |

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

select salary , last\_name 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.

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';



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

select last\_name, department\_id from employees where department\_id = 20 or department\_id = 50 order by last\_name;



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.

select last\_name as "EMPLOYEE" , salary as "MONTHLY SALARY" from employees where department\_id in (20,50) and salary between 5000 and 12000 order by last\_name;



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

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

select e.last\_name, d.dept\_name from employees e join department d on e.department\_id = d.dept\_id where not(dept\_name = 'manager');



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,commission\_pct desc;



1. Display the last name of all employees where the third letter of the name is a.

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.

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/.

SELECT e.last\_name,e.salary,d.dept\_name FROM employees e

join department d on e.department\_id = d.dept\_id

WHERE (dept\_name in ('stock clerk','sales representative')) and (salary not in(2500,3500,7000));



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 7** | | **USING SET OPERATORS** |
|  |  |
| **Date:** | 30/08/2024 |  |

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 dept\_id from department

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 c.country\_id, c.country\_name

FROM countries c

LEFT JOIN department d ON c.country\_id = d.country\_id WHERE d.country\_id IS NULL;



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 IN (10, 50, 20)

ORDER BY department\_id;



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 dept\_name, dept\_id FROM department;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 8** | | **WORKING WITH MULTIPLE TABLES** |
| **Date:** | 05/09/2024 |

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.dept\_name from employees e

join department d on e.department\_id = d.dept\_id;



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

select d.dept\_name,d.location\_id from department d

join employees e on d.dept\_id = e.department\_id where 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.dept\_name,d.location\_id,l.city from (department d inner join employees e on d.dept\_id = e.department\_id inner join location l on d.location\_id = l.location\_id) where 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.

select e.last\_name,d.dept\_name from department d

inner join employees e on d.dept\_id = e.department\_id where 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,d.dept\_name,e.department\_id from (department d inner join employees e on d.dept\_id = e.department\_id inner join location l on l.location\_id = d.location\_id) where 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 last\_name as "Employee",employee\_id as "Emp#",manager\_id as "Mgr#" from employees;



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

SELECT last\_name AS "Employee",employee\_id AS "Emp#",manager\_id AS "Mgr#" FROM employees ORDER BY 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 e.last\_name as "Employee",d.dept\_name as "department\_name",e.department\_id as "department\_no" from employees e

inner join department d on e.department\_id = d.dept\_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\_grade;

SELECT e.first\_name || ' ' || last\_name AS

"Employee",d.dept\_name,e.salary,g.grade\_level as "GRADE" FROM (employees e

inner join department d on e.department\_id = d.dept\_id inner join job\_grade g on e.department\_id = g.department\_id);



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 > '05-03-1986';



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 last\_name as "employee",hire\_date as "employee hired" FROM employees;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 9** | | **SUB QUERIES** |
| **Date:** | 06/09/2024 |

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 department\_id = ALL(

SELECT department\_id

FROM employees

WHERE last\_name = 'Zlotkey'

)

AND last\_name != 'Zlotkey';



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 EMPLOYEE\_ID, LAST\_NAME

FROM employees

WHERE DEPARTMENT\_ID IN (

SELECT DEPARTMENT\_ID

FROM employees

WHERE LAST\_NAME LIKE '%a%' and 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

INNER JOIN department d ON e.department\_id = d.dept\_id

WHERE e.department\_id IN (

SELECT dept\_id

FROM department

WHERE 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

WHERE e.manager\_id IN ( SELECT d.manager\_id

FROM department d

WHERE d.manager\_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 department d on e.department\_id = d.dept\_id

WHERE d.dept\_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 e.employee\_id, e.last\_name, e.salary

FROM employees e

WHERE e.salary > (

SELECT AVG(salary)

FROM employees

)

AND e.department\_id IN (

SELECT x.department\_id

FROM employees x

WHERE x.last\_name LIKE '%a%' AND x.last\_name LIKE '%u%'

);



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 10** | | **AGGREGATING DATA USING GROUP FUNCTIONS** |
|  |  |
| **Date:** | 12/09/2024 |  |

**Find the Solution for the following:**

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

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

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 - FALSE

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 ROUND(MAX(salary)) AS Maximum, ROUND(MIN(salary)) AS Minimum,

ROUND(SUM(salary)) AS Sum, ROUND(AVG(salary)) AS Average

FROM employees join department

on department.dept\_id = employees.department\_id group by dept\_name;



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 d.dept\_name , COUNT(\*) AS NumberOfEmployees

FROM Employees e

join department d on e.department\_id = d.dept\_id group by d.dept\_name;



1. Determine the number of managers without listing them. Label the column 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 "Lowest 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 EXTRACT(YEAR FROM hire\_date) AS "yearly wise employment", COUNT(\*)

FROM employees

GROUP BY EXTRACT(YEAR FROM hire\_date)

HAVING EXTRACT(YEAR FROM hire\_date) IN (1995, 1996, 1997, 1998);



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 d.dept\_name , sum(e.salary)

from employees e

join department d on e.department\_id = d.dept\_id where department\_id in (20,50,80,90) group by d.dept\_name;



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.dept\_name AS "Name", d.Location\_id AS "Location",

COUNT(e.department\_id) AS "Number of People", ROUND(AVG(e.Salary), 2) AS "Salary"

FROM department d

JOIN employees e ON d.dept\_id = e.department\_id

GROUP BY d.dept\_name, d.location\_id;



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 11** | | **PL SQL PROGRAMS** |
| **Date:** | 13/09/2024 |

**PROGRAM 1**

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

DECLARE

pl\_emp\_id employees.employee\_id%TYPE := 110; pl\_salary employees.salary%TYPE;

pl\_incentive NUMBER;

BEGIN

SELECT salary INTO pl\_salary

FROM employees

WHERE employee\_id = pl\_emp\_id;

pl\_incentive := pl\_salary \* 0.10;

UPDATE employees

SET incentive = pl\_incentive

WHERE employee\_id = pl\_emp\_id;

DBMS\_OUTPUT.PUT\_LINE('Incentive for employee ID ' || pl\_emp\_id || ' is ' || pl\_incentive);

COMMIT;

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 employeeName VARCHAR2(100);

"EmployeeID" NUMBER; BEGIN employeeName := 'John Doe';

"EmployeeID" := 40;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || employeeName);

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || "EmployeeID");

END;



**PROGRAM 3**

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

**DECLARE v\_employee\_id NUMBER := 122; v\_salary NUMBER; v\_new\_salary NUMBER;**

**v\_increase\_percentage NUMBER := 0.40;**

**BEGIN**

**SELECT salary INTO v\_salary**

**FROM employees**

**WHERE employee\_id = v\_employee\_id; v\_new\_salary := v\_salary + (v\_salary \* v\_increase\_percentage / 100);**

**UPDATE employees**

**SET salary = v\_new\_salary**

**WHERE employee\_id = v\_employee\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Employee ID ' || v\_employee\_id || ' new salary: ' || v\_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 check\_null is**

**value1 number := 10; value2 number := null;**

**begin if value1 is not null and value2 is null then**

**dbms\_output.put\_line('Both values are not null!!');**

**else dbms\_output.put\_line('Null value found');**

**end if;**

**end;**

**BEGIN check\_null;**

**END;**



**PROGRAM 5**

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

**declare**

**v\_employeename employees.first\_name%type; v\_employeeid NUMBER := 122;**

**begin**

**select first\_name into v\_employeename from employees where first\_name like '%e%' and employee\_id = v\_employeeid;**

**DBMS\_OUTPUT.PUT\_LINE(v\_employeename);**

**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 ab number :=10; cd number :=20; num\_small number; num\_large number;**

**begin if ab>cd then num\_small :=cd; num\_large :=ab; else num\_small :=ab; num\_large :=cd; end if;**

**dbms\_output.put\_line('small number = '||num\_small); dbms\_output.put\_line('large number = '||num\_large); End;**



**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 calculate\_incentive(p\_emp\_id employees.employee\_id%type, p\_target number) is**

**v\_incentive number(7,2); v\_salary employees.salary%type;**

**begin select salary into v\_salary from employees**

**where employee\_id = p\_emp\_id;**

**if p\_target >= 100000 then v\_incentive := v\_salary \* 0.1;**

**dbms\_output.put\_line('Incentive of ' || v\_incentive || ' calculated for employee ID ' ||**

**p\_emp\_id); else dbms\_output.put\_line('No incentive for employee ID ' || p\_emp\_id);**

**end if; End;**



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

**create or replace procedure incentive\_sale(p\_emp\_id employees.employee\_id%type, p\_sales number)**

**is**

**v\_incentive number(7,2);**

**begin if p\_sales > 100000 then v\_incentive := p\_sales \* 0.1;**

**elsif p\_sales between 50000 and 100000 then v\_incentive := p\_sales \* 0.05;**

**else v\_incentive := 0;**

**end if;**

**dbms\_output.put\_line('Incentive for employee ID ' || p\_emp\_id || ' is: ' || v\_incentive); End;**

**begin incentive\_sale(122,500000);**

**end;**



**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 no\_of\_emp number; vacancies number:=45; begin**

**select count(\*) into no\_of\_emp from employees where department\_id=50; if no\_of\_emp<vacancies then**

**dbms\_output.put\_line('vacancies are available'); else**

**dbms\_output.put\_line('vacancies are not available'); 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**

**v\_department\_id number := 55; v\_emp\_count number; v\_vacancies number := 50;**

**begin**

**select count(\*) into v\_emp\_count**

**from employees**

**where department\_id = v\_department\_id;**

**if v\_emp\_count < v\_vacancies then**

**dbms\_output.put\_line('Vacancies available: ' || (v\_vacancies - v\_emp\_count));**

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

**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 || ' ' || last\_name as name, job\_id, hire\_date,**

**salary from employees) loop dbms\_output.put\_line('ID: ' || i.employee\_id || ', Name: ' || i.name || ', Job: ' || i.job\_id**

**|| ', Hire Date: ' || i.hire\_date || ', Salary: ' || i.salary); end loop;**

**end;**



**PROGRAM 12**

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

**begin for i in (select e.employee\_id, e.first\_name || ' ' || e.last\_name as name, d.dept\_name from employees e**

**join department d on e.employee\_id = d.dept\_id) loop**

**dbms\_output.put\_line('ID: ' || i.employee\_id || ', Name: ' || i.name || ', Department: ' || i.dept\_name); end loop; End;**



**PROGRAM 13**

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

**begin for rec in (select e.employee\_id, d.dept\_name, min(salary) as min\_salary from**

**employees e join department d on e.employee\_ID = d.dept\_id group by e.employee\_id , d.dept\_name) loop dbms\_output.put\_line('Job ID: ' || rec.employee\_id || ', Title: ' || rec.dept\_name || ',**

**Min Salary: ' || rec.min\_salary); end loop; End;**



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

**begin for rec in (select e.employee\_id, d.dept\_name, min(salary) as min\_salary from**

**employees e join department d on e.employee\_ID = d.dept\_id group by e.employee\_id , d.dept\_name) loop dbms\_output.put\_line('Job ID: ' || rec.employee\_id || ', Title: ' || rec.dept\_name || ',**

**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.**

**Begin for rec in (select employee\_id, first\_name || ' ' || last\_name as name, hire\_date from employees) loop**

**dbms\_output.put\_line('ID: ' || rec.employee\_id || ', Name: ' || rec.name || ', Start Date: '**

**|| rec.hire\_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.**

**BEGIN**

**FOR rec IN (SELECT employee\_id, first\_name || ' ' || last\_name AS name, end\_date FROM employees) LOOP**

**dbms\_output.put\_line('ID: ' || rec.employee\_id ||**

**', Name: ' || rec.name ||**

**', End Date: ' ||**

**NVL(TO\_CHAR(rec.end\_date, 'YYYY-MM-DD'), 'Still Active')); END LOOP;**

**END;**



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 12** | | **PL SQL PROGRAMS** |
| **Date:** | 19/09/2024 |

**Program 1**

**FACTORIAL OF A NUMBER USING FUNCTION**

**DECLARE**

**n NUMBER := 10;**

**result NUMBER;**

**FUNCTION itfact(num NUMBER) RETURN NUMBER IS fact NUMBER := 1;**

**BEGIN**

**FOR i IN 1..num LOOP fact := fact \* i;**

**END LOOP;**

**RETURN fact;**

**END;**

**BEGIN**

**result := itfact(n);**

**DBMS\_OUTPUT.PUT\_LINE('The factorial of ' || n || ' is ' || result); END;**



**Program 2**

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

**CREATE OR REPLACE PROCEDURE book\_info( p\_book\_id IN NUMBER, p\_author OUT VARCHAR2, p\_title OUT VARCHAR2,**

**p\_published\_date OUT DATE**

**) AS**

**BEGIN**

**SELECT author, title, published\_date**

**INTO p\_author, p\_title, p\_published\_date**

**FROM books**

**WHERE book\_id = p\_book\_id;**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN p\_author := NULL; p\_title := NULL;**

**p\_published\_date := NULL;**

**WHEN OTHERS THEN**

**RAISE; END book\_info;**

**DECLARE v\_author VARCHAR2(100); v\_title VARCHAR2(100); v\_published\_date DATE; v\_book\_id NUMBER := 1; BEGIN book\_info(v\_book\_id, v\_author, v\_title, v\_published\_date);**

**IF v\_author IS NOT NULL THEN**

**DBMS\_OUTPUT.PUT\_LINE('Book ID: ' || v\_book\_id);**

**DBMS\_OUTPUT.PUT\_LINE('Author: ' || v\_author);**

**DBMS\_OUTPUT.PUT\_LINE('Title: ' || v\_title);**

**DBMS\_OUTPUT.PUT\_LINE('Published Date: ' || TO\_CHAR(v\_published\_date, 'YYYY- MM-DD'));**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('No book found with ID: ' || v\_book\_id); END IF;**

**END;**



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 13** | | **WORKING WITH TRIGGERS** |
| **Date:** | 20/09/2024 |

**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 employees**

**FOR EACH ROW**

**DECLARE pl\_dept\_count NUMBER; BEGIN**

**SELECT COUNT(\*)**

**INTO pl\_dept\_count**

**FROM department**

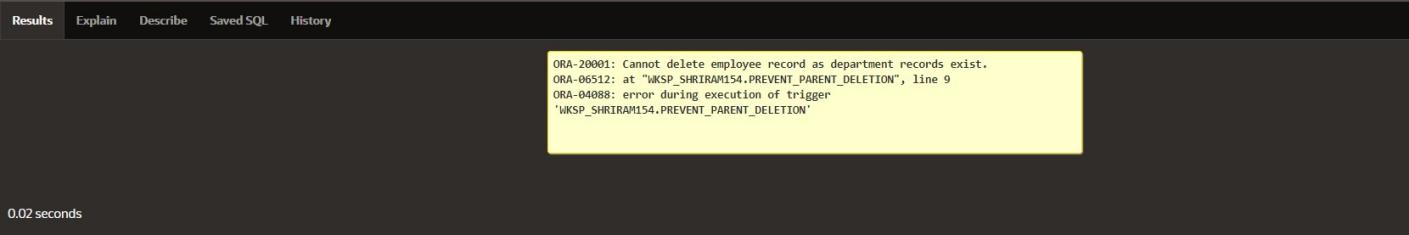
**WHERE dept\_id = :OLD.employee\_id;**

**IF pl\_dept\_count > 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Cannot delete employee record as department records exist.'); END IF; END;**

**DELETE FROM employees**

**WHERE employee\_id = 70;**



**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 prevent\_duplicate\_manager\_id**

**BEFORE INSERT OR UPDATE ON employees**

**FOR EACH ROW**

**DECLARE pl\_count NUMBER; BEGIN**

**SELECT COUNT(\*)**

**INTO pl\_count**

**FROM employees**

**WHERE manager\_id = :NEW.manager\_id**

**AND employee\_id != :NEW.employee\_id;**

**IF pl\_count > 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20003, 'Duplicate manager\_id found: ' ||**

**:NEW.manager\_id); END IF;**

**END;**

**INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number,**

**hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id)**

**VALUES (202, 'Jane', 'Smith',**

**'john006@gmail.com',7383922241,'11/9/2000','ST\_CLERK',10000,0.15,400,80);**



**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\_salary\_insertion**

**BEFORE INSERT ON employees**

**FOR EACH ROW**

**DECLARE**

**total\_salary NUMBER; threshold NUMBER := 100000;**

**BEGIN**

**SELECT SUM(salary)**

**INTO total\_salary**

**FROM employees;**

**IF (total\_salary + :NEW.salary) > threshold THEN**

**RAISE\_APPLICATION\_ERROR(-20004, 'Insertion denied: Total salary exceeds the threshold of ' || threshold); END IF;**

**END;**

**INSERT INTO employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id)**

**VALUES (203, 'Charlie', 'Brown', 'charlie203@gmail.com', '9122334455','03/01/2021', '#cb203', 5000, 0.20, 1000, 50);**



**PROGRAM 4**

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

**CREATE OR REPLACE TRIGGER audit\_changes**

**AFTER UPDATE OF salary, job\_id ON employees**

**FOR EACH ROW**

**BEGIN**

**IF :OLD.salary != :NEW.salary OR :OLD.job\_id != :NEW.job\_id THEN**

**INSERT INTO employee\_audit ( employee\_id, old\_salary, new\_salary, old\_job\_title, new\_job\_title, change\_timestamp, changed\_by**

**) VALUES (**

**:OLD.employee\_id,**

**:OLD.salary,**

**:NEW.salary,**

**:OLD.job\_id,**

**:NEW.job\_id,**

**SYSTIMESTAMP,**

**USER**

**);**

**END IF;**

**END;**

**UPDATE employees**

**SET salary = 55000, job\_id = 'ST\_CLERK'**

**WHERE employee\_id = 176;**

**SELECT \* FROM employee\_audit;**



**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 trg\_audit\_employees**

**AFTER INSERT OR UPDATE OR DELETE ON employees**

**FOR EACH ROW DECLARE**

**v\_old\_values CLOB; v\_new\_values CLOB;**

**BEGIN**

**IF INSERTING THEN v\_old\_values := NULL;**

**v\_new\_values := 'employee\_id: ' || :NEW.employee\_id || ', ' ||**

**'first\_name: ' || :NEW.first\_name || ', ' ||**

**'salary: ' || :NEW.salary;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, new\_values) VALUES ('INSERT', 'employees', :NEW.employee\_id, USER, v\_new\_values);**

**ELSIF UPDATING THEN**

**v\_old\_values := 'employee\_id: ' || :OLD.employee\_id || ', ' ||**

**'first\_name: ' || :OLD.first\_name || ', ' ||**

**'salary: ' || :OLD.salary;**

**v\_new\_values := 'employee\_id: ' || :NEW.employee\_id || ', ' ||**

**'first\_name: ' || :NEW.first\_name || ', ' ||**

**'salary: ' || :NEW.salary;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, old\_values, new\_values)**

**VALUES ('UPDATE', 'employees', :NEW.employee\_id, USER, v\_old\_values, v\_new\_values);**

**ELSIF DELETING THEN**

**v\_old\_values := 'employee\_id: ' || :OLD.employee\_id || ', ' ||**

**'first\_name: ' || :OLD.first\_name || ', ' ||**

**'salary: ' || :OLD.salary;**

**v\_new\_values := NULL;**

**INSERT INTO audit\_log (action, table\_name, record\_id, changed\_by, old\_values)**

**VALUES ('DELETE', 'employees', :OLD.employee\_id, USER, v\_old\_values);**

**END IF;**

**END trg\_audit\_employees;**

**INSERT INTO employees (employee\_id, first\_name, salary) VALUES (3, 'Ball', 50000);**



**UPDATE employees**

**SET salary = 55000 WHERE employee\_id = 3;**



**DELETE FROM employees**

**WHERE employee\_id = 3;**

**SELECT \* FROM audit\_log;**



**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 TABLE transactions ( transaction\_id NUMBER PRIMARY KEY,**

**amount NUMBER,**

**running\_total NUMBER**

**);**

**CREATE OR REPLACE TRIGGER update\_running\_total**

**FOR INSERT ON transactions COMPOUND TRIGGER**

**TYPE amount\_array IS TABLE OF NUMBER INDEX BY PLS\_INTEGER; new\_amounts amount\_array;**

**BEFORE EACH ROW IS**

**BEGIN new\_amounts(:NEW.transaction\_id) := :NEW.amount; END BEFORE EACH ROW;**

**AFTER STATEMENT IS**

**BEGIN**

**DECLARE v\_total NUMBER;**

**BEGIN**

**SELECT NVL(MAX(running\_total), 0)**

**INTO v\_total FROM transactions;**

**FOR i IN new\_amounts.FIRST .. new\_amounts.LAST LOOP v\_total := v\_total + new\_amounts(i);**

**UPDATE transactions**

**SET running\_total = v\_total**

**WHERE transaction\_id = i;**

**END LOOP;**

**END;**

**END AFTER STATEMENT;**

**END update\_running\_total;**

**INSERT INTO transactions (transaction\_id, amount)**

**VALUES (1, 10000);**

**INSERT INTO transactions (transaction\_id, amount)**

**VALUES (2, 20000);**



**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 TABLE inventory ( item\_id NUMBER PRIMARY KEY, item\_name VARCHAR2(100), stock\_level NUMBER**

**);**

**CREATE TABLE orders ( order\_id NUMBER PRIMARY KEY, item\_id NUMBER, quantity NUMBER, order\_status VARCHAR2(20),**

**CONSTRAINT fk\_item FOREIGN KEY (item\_id) REFERENCES inventory(item\_id)**

**);**

**CREATE OR REPLACE TRIGGER validate\_stock\_before\_order**

**BEFORE INSERT ON orders**

**FOR EACH ROW**

**DECLARE v\_stock\_level NUMBER; v\_pending\_orders NUMBER;**

**BEGIN**

**SELECT stock\_level**

**INTO v\_stock\_level**

**FROM inventory**

**WHERE item\_id = :NEW.item\_id;**

**SELECT NVL(SUM(quantity), 0)**

**INTO v\_pending\_orders**

**FROM orders**

**WHERE item\_id = :NEW.item\_id AND order\_status = 'Pending';**

**IF (:NEW.quantity + v\_pending\_orders) > v\_stock\_level THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient stock for item: ' || :NEW.item\_id);**

**END IF;**

**END;**

**INSERT INTO orders (order\_id, item\_id, quantity, order\_status) VALUES (1, 101, 5, 'Pending');**



**INSERT INTO orders (order\_id, item\_id, quantity, order\_status) VALUES (2, 103, 20, 'Pending');**







|  |  |  |
| --- | --- | --- |
| **Ex.No.: 14** | | **MONGO DB** |
| **Date:** | 26/09/2024 |

**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", "Chinees"] } },**

**{ name: { $regex: /^Wil/i } }**

**]**

**},**

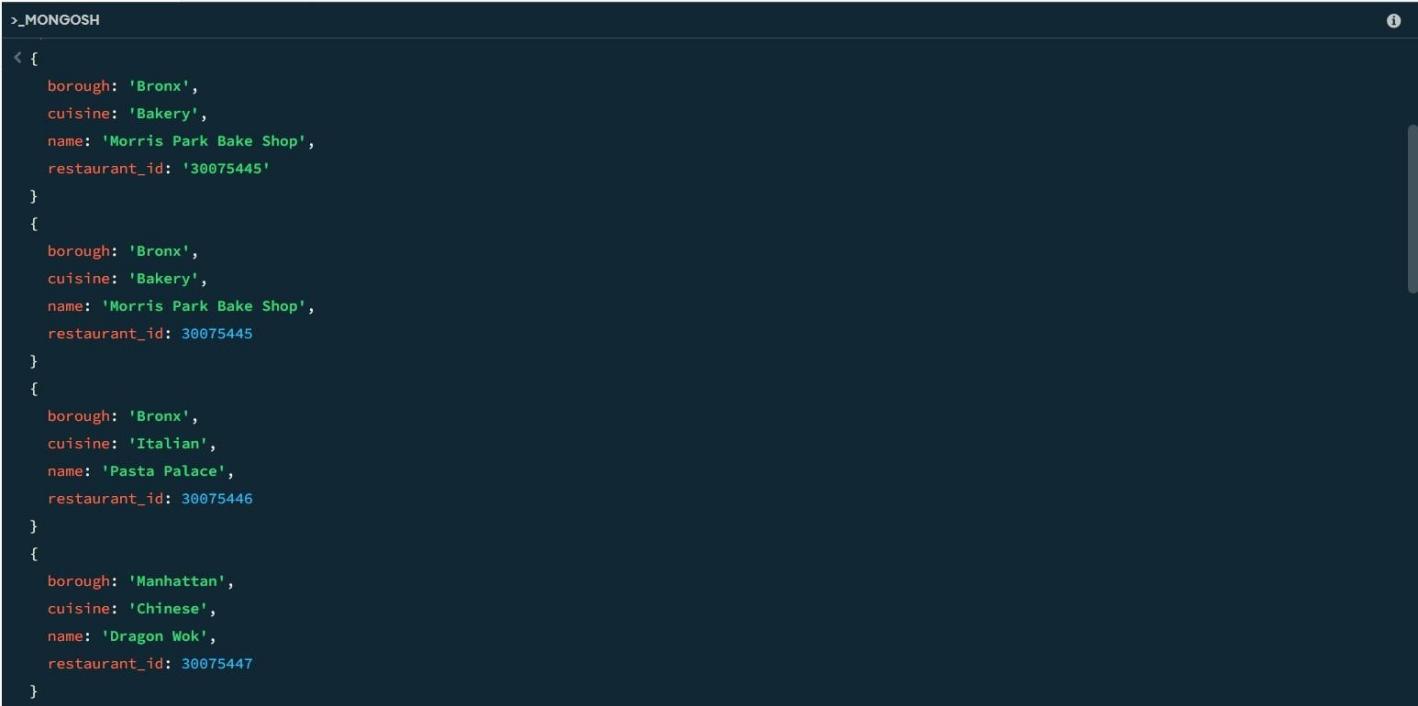
**{**

**restaurant\_id: 1, name: 1, borough: 1, cuisine: 1,**

**\_id: 0**

**}**

**);**



**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**

**}**

**}**

**},**

**{**

**restaurant\_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(**

**{**

**"grades.1": {**

**$elemMatch: { grade: "A", score: 9**

**}**

**}**

**},**

**{**

**restaurant\_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,**

**\_id: 0**

**}**

**);**

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

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

**SAMPLE OUTPUT:-**

**{**

**\_id: ObjectId('671b5e6d56ec9972ca8f5dc4'), address: { building: 5566, coord: [ -73.867377,**

**40.854047**

**],**

**street: '28th Avenue',**

**zipcode: 10490**

**},**

**borough: 'Bronx', cuisine: 'BBQ', grades: [**

**{**

**date: 2014-03-03T00:00:00.028Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2013-09-11T00:00:00.028Z,**

**grade: 'A', score: 7**

**},**

**{**

**date: 2013-01-24T00:00:00.028Z,**

**grade: 'A',**

**score: 11**

**},**

**{**

**date: 2011-11-23T00:00:00.028Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.028Z, grade: 'B',**

**score: 15**

**}**

**],**

**name: 'BBQ Haven',**

**restaurant\_id: 30075473**

**}**

**{**

**\_id: ObjectId('671b5dab56ec9972ca8f5db0'), address: { building: 5566, coord: [ -73.859377,**

**40.850047**

**],**

**street: '8th Avenue', zipcode: 10470**

**},**

**borough: 'Manhattan', cuisine: 'French',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.008Z,**

**grade: 'A',**

**score: 7**

**},**

**{**

**date: 2013-09-11T00:00:00.008Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2013-01-24T00:00:00.008Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.008Z,**

**grade: 'B',**

**score: 15**

**},**

**{**

**date: 2011-03-10T00:00:00.008Z,**

**grade: 'A',**

**score: 6**

**}**

**],**

**name: 'Bistro Belle',**

**restaurant\_id: 30075453**

**}**

**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 });**

## SAMPLE OUTPUT

**{**

**\_id: ObjectId('671b5e9456ec9972ca8f5dc8'), address: { building: 9900, coord: [ -73.868977,**

**40.854847**

**],**

**street: '32nd Avenue',**

**zipcode: 10494**

**},**

**borough: 'Manhattan', cuisine: 'Russian', grades: [**

**{**

**date: 2014-03-03T00:00:00.032Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2013-09-11T00:00:00.032Z,**

**grade: 'B',**

**score: 5**

**},**

**{**

**date: 2013-01-24T00:00:00.032Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-11-23T00:00:00.032Z,**

**grade: 'A',**

**score: 8**

**},**

**{**

**date: 2011-03-10T00:00:00.032Z,**

**grade: 'A',**

**score: 11**

**}**

**],**

**name: "Tsar's Table",**

**restaurant\_id: 30075477**

**}**

**{**

**\_id: ObjectId('671b5e6d56ec9972ca8f5dbe'), address: { building: 9900, coord: [ -73.864977,**

**40.852847**

**],**

**street: '22nd Avenue',**

**zipcode: 10484**

**},**

**borough: 'Bronx', cuisine: 'Italian',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.022Z,**

**grade: 'A',**

**score: 8**

**},**

**{**

**date: 2013-09-11T00:00:00.022Z,**

**grade: 'B',**

**score: 5**

**},**

**{**

**date: 2013-01-24T00:00:00.022Z,**

**grade: 'A', score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.022Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.022Z,**

**grade: 'A',**

**score: 14**

**}**

**],**

**name: 'Trattoria Bella',**

**restaurant\_id: 30075467**

**}**

**7. 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 });**

**SAMPLE OUTPUT:-**

**{**

**\_id: ObjectId('671b5d549d3d63480e0a64e9'), address: { building: 2233, coord: [ -73.858177,**

**40.849447**

**],**

**street: '5th Avenue', zipcode: 10467**

**},**

**borough: 'Bronx', cuisine: 'American', grades: [**

**{**

**date: 2014-03-03T00:00:00.005Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2013-09-11T00:00:00.005Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.005Z,**

**grade: 'B',**

**score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.005Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.005Z,**

**grade: 'A',**

**score: 14**

**}**

**],**

**name: 'Burger Bistro',**

**restaurant\_id: 30075450**

**}**

**{**

**\_id: ObjectId('671b5e6d56ec9972ca8f5dc4'), address: { building: 5566, coord: [ -73.867377,**

**40.854047**

**],**

**street: '28th Avenue', zipcode: 10490**

**},**

**borough: 'Bronx', cuisine: 'BBQ', grades: [**

**{**

**date: 2014-03-03T00:00:00.028Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2013-09-11T00:00:00.028Z,**

**grade: 'A', score: 7**

**},**

**{**

**date: 2013-01-24T00:00:00.028Z,**

**grade: 'A',**

**score: 11**

**},**

**{**

**date: 2011-11-23T00:00:00.028Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.028Z,**

**grade: 'B', score: 15**

**}**

**],**

**name: 'BBQ Haven',**

**restaurant\_id: 30075473**

**}**

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(**

**{**

**"address.coord": { $type: "double" }**

**}**

**);**

**SAMPLE OUTPUT:-**

**{**

**\_id: ObjectId('671b92d339ec8a9bc8b6588b'), address: { building: '1007', coord: [ -73.856077,**

**40.848447**

**],**

**street: 'Morris Park Ave', zipcode: '10462'**

**},**

**borough: 'Bronx', cuisine: 'Bakery', grades: [**

**{**

**date: 2014-03-03T00:00:00.000Z,**

**grade: 'A',**

**score: 2**

**},**

**{**

**date: 2013-09-11T00:00:00.000Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.000Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.000Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.000Z,**

**grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Morris Park Bake Shop',**

**restaurant\_id: '30075445'**

**}**

**{**

**\_id: ObjectId('671b5d549d3d63480e0a64e5'),**

**address: { building: 1234, coord: [ -73.856577,**

**40.848647**

**],**

**street: '1st Avenue',**

**zipcode: 10463**

**},**

**borough: 'Bronx', cuisine: 'Italian',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.001Z,**

**grade: 'A',**

**score: 5**

**},**

**{**

**date: 2013-09-11T00:00:00.001Z,**

**grade: 'A', score: 8**

**},**

**{**

**date: 2013-01-24T00:00:00.001Z,**

**grade: 'B',**

**score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.001Z,**

**grade: 'A',**

**score: 7**

**},**

**{**

**date: 2011-03-10T00:00:00.001Z,**

**grade: 'A', score: 15**

**}**

**],**

**name: 'Pasta Palace',**

**restaurant\_id: 30075446**

**}**

**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] }**

**},**

**{**

**restaurant\_id: 1, name: 1, grades: 1,**

**\_id: 0**

**}**

**);**

## SAMPLE OUTPUT:-

**{**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.000Z,**

**grade: 'A',**

**score: 2**

**},**

**{**

**date: 2013-09-11T00:00:00.000Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.000Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.000Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.000Z,**

**grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Morris Park Bake Shop',**

**restaurant\_id: '30075445'**

**}**

**{**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.001Z,**

**grade: 'A',**

**score: 5**

**},**

**{**

**date: 2013-09-11T00:00:00.001Z,**

**grade: 'A',**

**score: 8**

**},**

**{**

**date: 2013-01-24T00:00:00.001Z,**

**grade: 'B',**

**score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.001Z,**

**grade: 'A', score: 7**

**},**

**{**

**date: 2011-03-10T00:00:00.001Z,**

**grade: 'A',**

**score: 15**

**}**

**],**

**name: 'Pasta Palace', restaurant\_id: 30075446**

**}**

**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.0": 1, // Longitude "address.coord.1": 1, // Latitude 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.0": 1, // Longitude "address.coord.1": 1, // Latitude 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 }**

**}**

**);**

## SAMPLE OUTPUT:-

**{**

**\_id: ObjectId('671b92d339ec8a9bc8b6588b'), address: {**

**building: '1007',**

**coord: [**

**-73.856077,**

**40.848447**

**],**

**street: 'Morris Park Ave',**

**zipcode: '10462'**

**},**

**borough: 'Bronx', cuisine: 'Bakery', grades: [**

**{**

**date: 2014-03-03T00:00:00.000Z,**

**grade: 'A',**

**score: 2**

**},**

**{**

**date: 2013-09-11T00:00:00.000Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.000Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.000Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.000Z,**

**grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Morris Park Bake Shop',**

**restaurant\_id: '30075445'**

**}**

**{**

**\_id: ObjectId('671b5d549d3d63480e0a64e6'),**

**address: {**

**building: 5678, coord: [ -73.856977,**

**40.848847**

**],**

**street: '2nd Avenue', zipcode: 10464**

**},**

**borough: 'Manhattan', cuisine: 'Chinese',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.002Z, grade: 'B',**

**score: 4**

**},**

**{**

**date: 2013-09-11T00:00:00.002Z, grade: 'A',**

**score: 9**

**},**

**{**

**date: 2013-01-24T00:00:00.002Z, grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.002Z, grade: 'A',**

**score: 8**

**},**

**{**

**date: 2011-03-10T00:00:00.002Z, grade: 'B',**

**score: 16**

**}**

**],**

**name: 'Dragon Wok', restaurant\_id: 30075447**

**}**

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.**

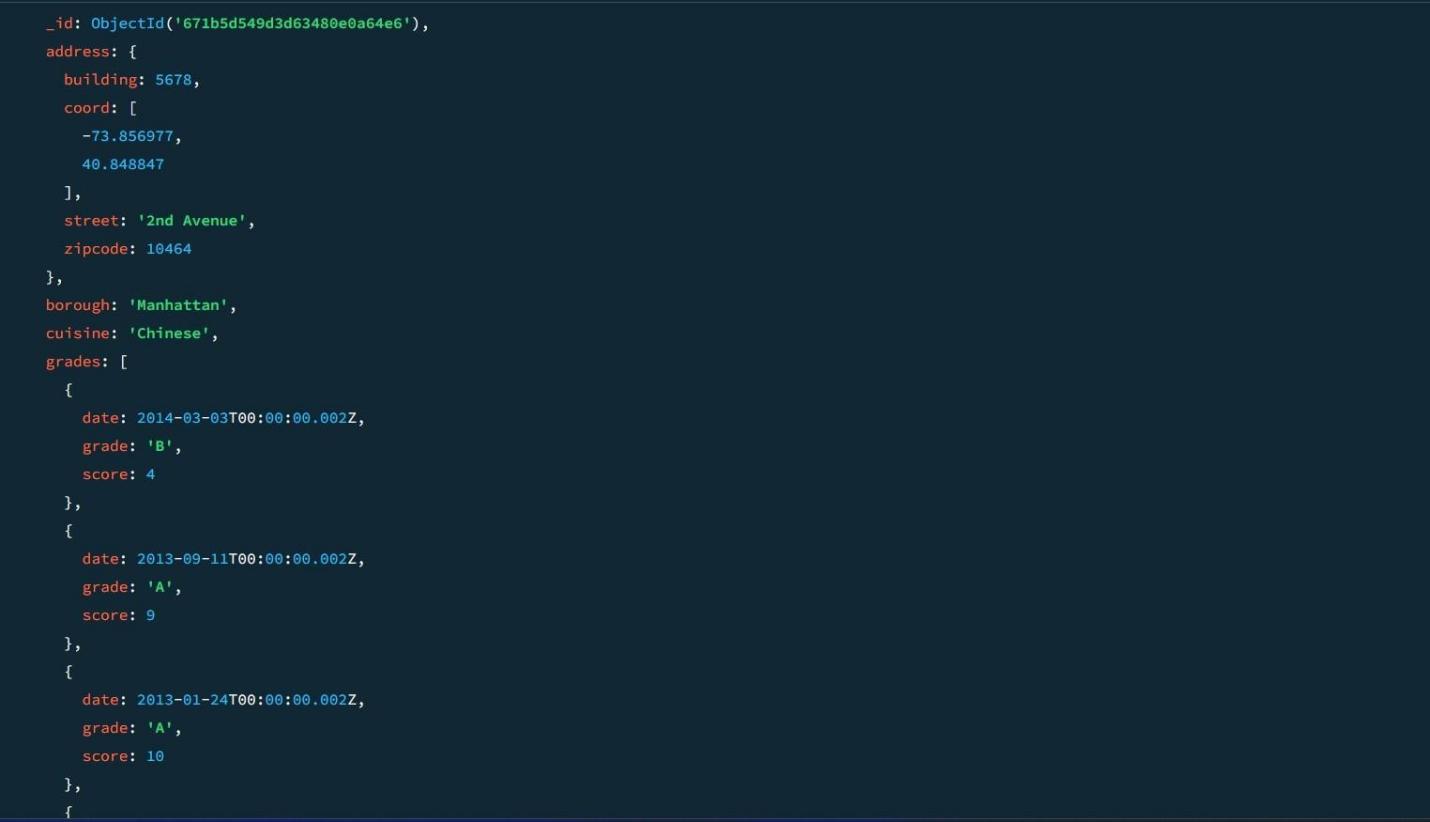
**db.restaurants.find(**

**{**

**"grades.score": { $lt: 5 }, borough: "Manhattan"**

**}**

**);**



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.**

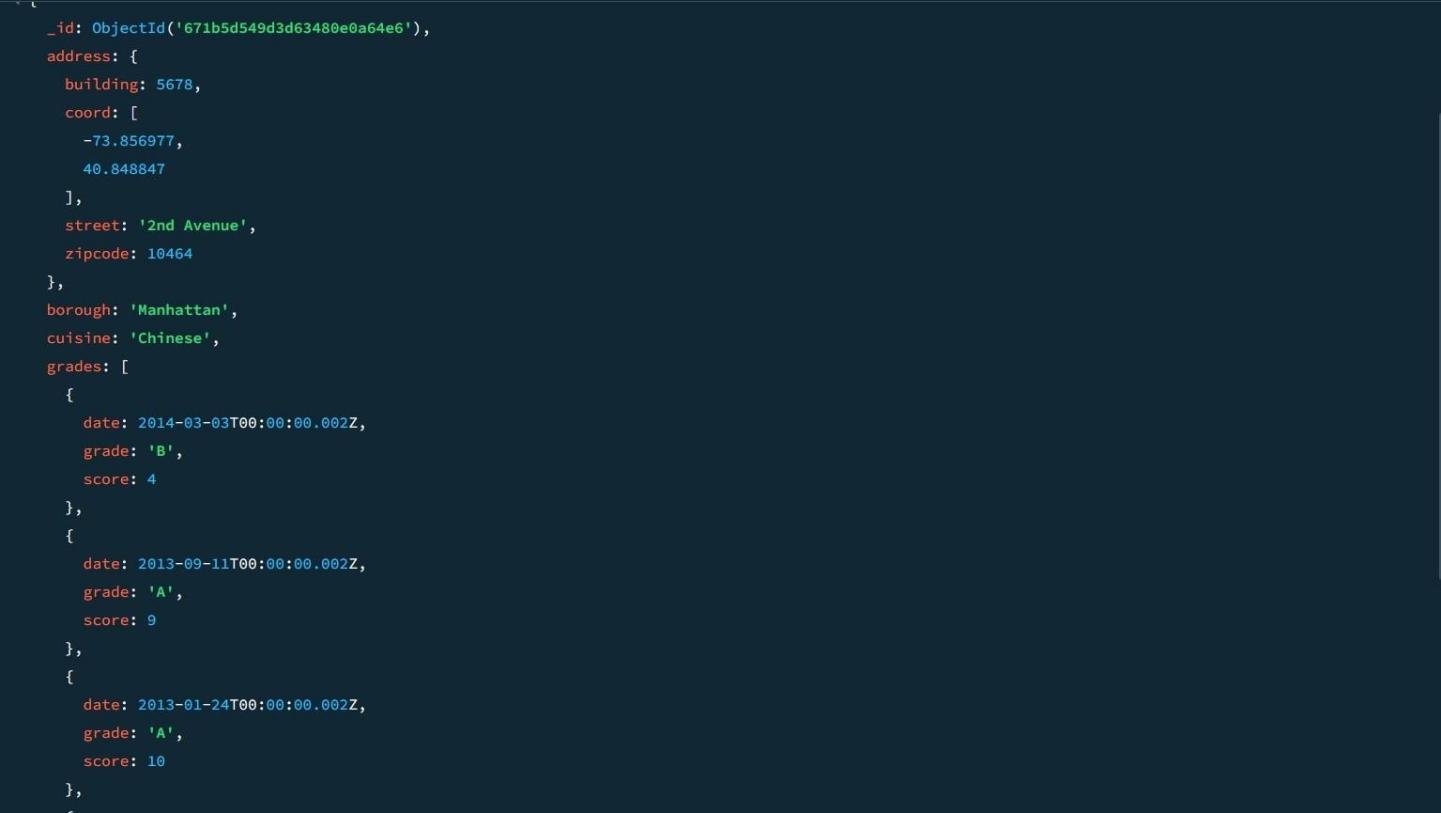
**db.restaurants.find(**

**{**

**"grades.score": { $lt: 5 }, borough: { $in: ["Manhattan", "Brooklyn"] }**

**}**

**);**



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, and their cuisine is not American.**

**db.restaurants.find(**

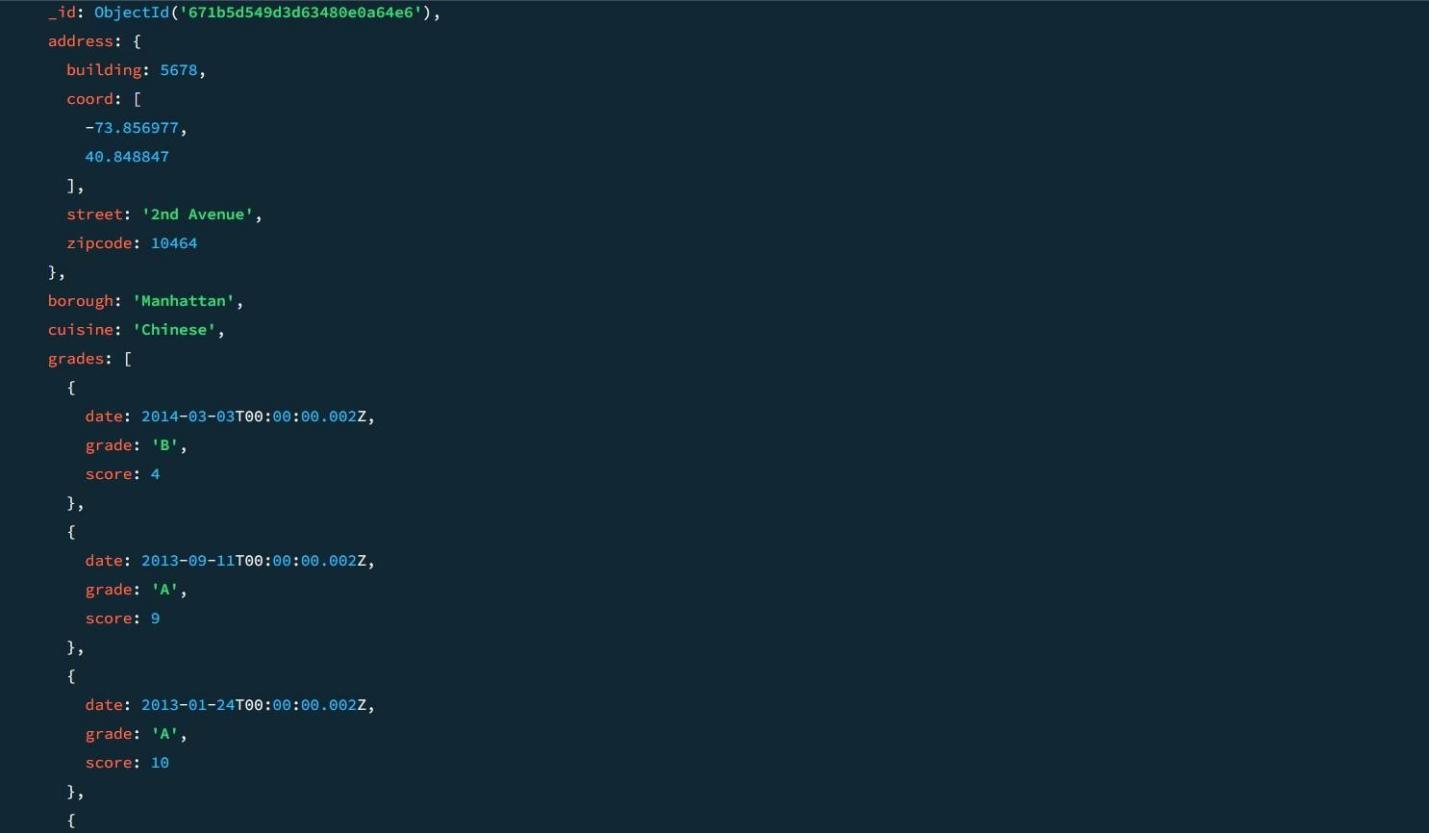
**{**

**"grades.score": { $lt: 5 },**

**borough: { $in: ["Manhattan", "Brooklyn"] }, cuisine: { $ne: "American" }**

**}**

**);**



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, and their cuisine is not American or Chinese.**

**db.restaurants.find(**

**{**

**"grades.score": { $lt: 5 }, borough: { $in: ["Manhattan", "Brooklyn"] }, cuisine: { $nin: ["American", "Chinese"] }**

**}**

**);**

1. **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 } }**

**]**

**}**

**}**

**);**

## SAMPLE OUTPUT:-

**{**

**\_id: ObjectId('671b92d339ec8a9bc8b6588b'),**

**address: { building: '1007', coord: [ -73.856077,**

**40.848447**

**],**

**street: 'Morris Park Ave', zipcode: '10462'**

**},**

**borough: 'Bronx', cuisine: 'Bakery',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.000Z,**

**grade: 'A',**

**score: 2**

**},**

**{**

**date: 2013-09-11T00:00:00.000Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.000Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.000Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.000Z,**

**grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Morris Park Bake Shop',**

**restaurant\_id: '30075445'**

**}**

**{**

**\_id: ObjectId('671b5c5f9d3d63480e0a64e4'), address: { building: 1007, coord: [ -73.856077,**

**40.848447**

**],**

**street: 'Morris Park Ave',**

**zipcode: 10462**

**},**

**borough: 'Bronx', cuisine: 'Bakery', grades: [**

**{**

**date: 2014-03-03T00:00:00.000Z,**

**grade: 'A',**

**score: 2**

**},**

**{**

**date: 2013-09-11T00:00:00.000Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.000Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2011-11-23T00:00:00.000Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.000Z,**

**grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Morris Park Bake Shop', restaurant\_id: 30075445**

**}**

**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: { $in: ["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 } }**

**]**

**},**

**cuisine: { $ne: "American" }**

**}**

**);**

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

**db.restaurants.find(**

**{**

**borough: { $in: ["Manhattan", "Brooklyn"] }, grades: {**

**$all: [**

**{ $elemMatch: { score: 2 } },**

**{ $elemMatch: { score: 6 } }**

**]**

**},**

**cuisine: { $nin: ["American", "Chinese"] }**

**}**

**);**

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

**db.restaurants.find(**

**{**

**$or: [**

**{ "grades.score": 2 },**

**{ "grades.score": 6 }**

**]**

**}**

**);**

## SAMPLE OUTPUT:-

**{**

**\_id: ObjectId('671b5d549d3d63480e0a64e9'), address: { building: 2233, coord: [ -73.858177,**

**40.849447**

**],**

**street: '5th Avenue', zipcode: 10467**

**},**

**borough: 'Bronx', cuisine: 'American', grades: [**

**{**

**date: 2014-03-03T00:00:00.005Z,**

**grade: 'A',**

**score: 10**

**},**

**{**

**date: 2013-09-11T00:00:00.005Z,**

**grade: 'A',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.005Z,**

**grade: 'B',**

**score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.005Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2011-03-10T00:00:00.005Z,**

**grade: 'A',**

**score: 14**

**}**

**],**

**name: 'Burger Bistro', restaurant\_id: 30075450**

**}**

**{**

**\_id: ObjectId('671b5dab56ec9972ca8f5daf'), address: { building: 4455, coord: [ -73.858977,**

**40.849847**

**],**

**street: '7th Avenue', zipcode: 10469**

**},**

**borough: 'Bronx', cuisine: 'Thai',**

**grades: [**

**{**

**date: 2014-03-03T00:00:00.007Z,**

**grade: 'A',**

**score: 9**

**},**

**{**

**date: 2013-09-11T00:00:00.007Z,**

**grade: 'B',**

**score: 6**

**},**

**{**

**date: 2013-01-24T00:00:00.007Z,**

**grade: 'A',**

**score: 12**

**},**

**{**

**date: 2011-11-23T00:00:00.007Z, grade: 'A',**

**score: 8**

**},**

**{**

**date: 2011-03-10T00:00:00.007Z, grade: 'B',**

**score: 14**

**}**

**],**

**name: 'Thai Delight', restaurant\_id: 30075452**

**}**

**MOVIES COLLECTION**

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

**db.movies.find({ year: 1893 });**

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 } });**

**SAMPLE OUTPUT:-**

**{**

**\_id: ObjectId('573a1390f29313caabcd42ec'),**

**plot: 'An astronaut stranded on Mars must survive alone.', genres: [**

**'Sci-Fi',**

**'Drama'**

**],**

**runtime: 135, cast: [**

**'Matt Damon',**

**'Jessica Chastain'**

**],**

**poster: 'https://m.media-amazon.com/images/poster4.jpg', title: 'Mars Alone', fullplot: 'An astronaut, left alone on Mars, struggles to survive with**

**limited resources while awaiting rescue.',**

**languages: [**

**'English'**

**],**

**released: 2015-10-02T00:00:00.000Z,**

**directors: [**

**'Ridley Scott'**

**],**

**rated: 'PG-13', awards: { wins: 8,**

**nominations: 6, text: '8 wins & 6 nominations.'**

**},**

**lastupdated: '2021-08-09 17:22:30.000000000', year: 2015, imdb: { rating: 8, votes: 25650,**

**id: 443**

**},**

**countries: [ 'USA'**

**],**

**type: 'movie', tomatoes: { viewer: {**

**rating: 4.5,**

**numReviews: 2201,**

**meter: 93**

**},**

**fresh: 18, critic: {**

**rating: 8.5,**

**numReviews: 25,**

**meter: 96**

**},**

**rotten: 1, lastUpdated: 2021-07-19T21:20:55.000Z**

**}**

**}**

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

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

**SAMPLE OUTPUT:-**

**{**

**\_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/MV5BMTU3NjE5NzYtYTYyNS00MDVmLWIwYjg**

**tMmYwYWIxZDYyNzU2XkEyXkFqcGdeQXVyNzQzNzQxNzI@.\_V1\_SY1**

**000\_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: 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: 2015-08-08T19:16:10.000Z**

**}**

**}**

**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" });**

1. **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" });**



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

**db.movies.find({ rated: "UNRATED" });**

1. **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 } });**



1. **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 } });**



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.rating": { $gt: 4 } });**



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

**db.movies.find({ "awards.wins": { $gt: 0 } });**



1. **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(**

**{ "awards.nominations": { $gt: 0 } },**

**{ title: 1,**

**languages: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1**

**}**

**);**



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

**db.movies.find(**

**{ cast: "Charles Kayser" },**

**{ title: 1,**

**languages: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1,**

**countries: 1**

**}**

**);**

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

**db.movies.find(**

**{ released: ISODate("1893-05-09T00:00:00Z") },**

**{ title: 1,**

**languages: 1, released: 1, directors: 1, writers: 1,**

**countries: 1**

**}**

**);**

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

**db.movies.find(**

**{ title: { $regex: /scene/i } },**

**{ title: 1,**

**languages: 1,**

**released: 1, directors: 1, writers: 1,**

**countries: 1**

**}**

**);**

|  |  |  |
| --- | --- | --- |
| **Ex.No.: 15** | | **OTHER DATABASE OBJECTS** |
| **Date:** | 27/09/2024 |

**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\_SEQ**

**START WITH 200**

**INCREMENT BY 10**

**MAXVALUE 1000**

**NOCACHE**

**NOCYCLE;**

**2. 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,**

**LAST\_NUMBER FROM USER\_SEQUENCES;**



**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.**

**INSERT INTO DEPT (DEPT\_ID, DEPT\_NAME)**

**VALUES (DEPT\_ID\_SEQ.NEXTVAL, 'Education');**

**INSERT INTO DEPT (DEPT\_ID, DEPT\_NAME)**

**VALUES (DEPT\_ID\_SEQ.NEXTVAL, 'Administration');**

**SELECT \* FROM DEPT**

**WHERE DEPT\_NAME IN ('Education', 'Administration');**



1. **Create a non unique index on the foreign key column (DEPARTMENT\_ID) in the EMPLOYEES table.**

**CREATE INDEX employees\_department\_id\_idx**

**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';**



|  |  |  |
| --- | --- | --- |
| **Ex.No.: 16** | | **CONTROLLING USER ACCESS** |
| **Date:** | 03/10/2024 |

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 a user should be given to log on to the Oracle Server is the CREATE SESSION privilege.**

**Type of Privilege: This is a system privilege.**

**GRANT CREATE SESSION TO username;**

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

**the user needs the CREATE TABLE privilege.**

**The CREATE TABLE privilege allows the user to create new tables in their own schema.**

**GRANT CREATE TABLE TO username;**

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

**When you create a table, only you as the table owner (or a user with the ADMIN OPTION or GRANT ANY PRIVILEGE system privilege) can grant privileges on your table to other users.**

**GRANT SELECT ON your\_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?**

**As a DBA, to simplify the process of granting the same system privileges to multiple users, you should use roles.**

**CREATE ROLE my\_role;**

**GRANT CREATE SESSION TO my\_role;**

**GRANT CREATE TABLE TO my\_role;**

**GRANT my\_role TO user1;**

**GRANT my\_role TO user2;**

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 Access to Your DEPARTMENTS Table**

**GRANT SELECT ON your\_username.DEPARTMENTS TO other\_user;**

**Grant Query Access to Other User's DEPARTMENTS Table**

**GRANT SELECT ON other\_user.DEPARTMENTS TO your\_username;**

1. **Query all the rows in your DEPARTMENTS table.**

**SELECT \* FROM DEPARTMENT;**



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.**

**INSERT INTO DEPARTMENT(dept\_id,**

**DEPT\_NAME,manager\_id,location\_id,country\_id,manager\_name)**

**VALUES (500, 'Education',300,12,'BAN','ball');**

**INSERT INTO DEPARTMENT(dept\_id,**

**DEPT\_NAME,manager\_id,location\_id,country\_id,manager\_name) VALUES (510, 'Human Resources',150,10,'AUS','john');**

**SELECT \* FROM DEPARTMENT;**



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.**

**REVOKE SELECT ON team1\_user.DEPARTMENTS FROM other\_user;**

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

**DELETE FROM DEPARTMENT**

**WHERE DEPT\_ID IN (500, 510);**