

RAJALAKSHMI ENGINEERING COLLEGE
RAJALAKSHMI NAGAR, THANDALAM – 602 105



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

**CS19443 DATABASE MANAGEMENT
SYSTEMS LAB**

LABORATORY RECORD NOTE BOOK
Comprehensive SQL Query Execution Archive

Name : **SAIVISHWARAM R**

Year / Branch / Section : **II / B.E CSE / D**

University Register No. : **2116220701239**

College Roll No. : **220701239**

Semester : **IV**

Academic Year : **2023-2024**

CREATING AND MANAGING TABLES

EX.NO:1

DATE: 15-2-24

REG.NO: 220701239

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

QUERY:

```
CREATE TABLE DEPT (Dept_ID number(6) not null, Dept_name varchar(20), Manager_ID varchar(6), Location_ID number(4));
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. The main area contains a command line with the following text:

```
1 CREATE TABLE DEPT (Dept_ID number(6) not null, Dept_name varchar(20), Manager_ID varchar(6), Location_ID number(4));
```

Below the command line, there is a results tab labeled "Results". The output section displays the message "Table created." and "0.03 seconds".

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

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

QUERY:

```
CREATE TABLE EMP( ID int, Last_name varchar(25), First_name varchar(25),
Dept_ID number(7));
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. Below the toolbar, the SQL command is entered:

```
1 CREATE TABLE EMP( ID int, Last_name varchar(25), First_name varchar(25),
Dept_ID number(7));
```

At the bottom, the Results tab is selected. The output shows:

```
Table created.  
0.03 seconds
```

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

QUERY:

```
ALTER TABLE EMP MODIFY(Last_name varchar(50));
```

OUTPUT:

The screenshot shows a MySQL command-line interface. The top bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and a 'Run' button. Below the bar, the command `ALTER TABLE EMP MODIFY(Last_name varchar(50));` is entered. The results section at the bottom displays the message `Table altered.` and a execution time of `0.05 seconds`.

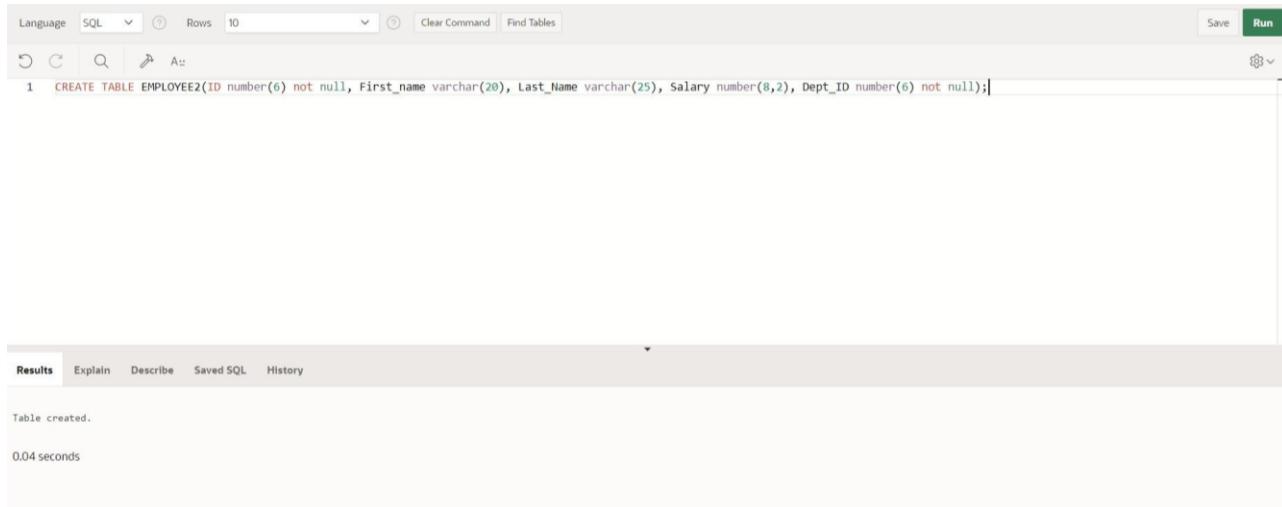
Results	Explain	Describe	Saved SQL	History
Table altered. 0.05 seconds				

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

QUERY:

```
CREATE TABLE EMPLOYEE2(ID number(6) not null, First_name varchar(20),  
Last_Name varchar(25), Salary number(8,2), Dept_ID number(6) not null);
```

OUTPUT:



The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (set to SQL), Rows (set to 10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. Below the toolbar, the SQL command is entered:

```
1 CREATE TABLE EMPLOYEE2(ID number(6) not null, First_name varchar(20), Last_Name varchar(25), Salary number(8,2), Dept_ID number(6) not null);
```

Below the command, the Results tab is selected. The output shows:

```
Table created.  
0.04 seconds
```

5. Drop the EMP table.

QUERY:

```
DROP TABLE EMP;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Save, and Run. Below the toolbar, the query is entered: `1 DROP TABLE EMP`. The main area displays the results tab, which shows the output of the query: `Table dropped.` and a execution time of `0.06 seconds`.

Results	Explain	Describe	Saved SQL	History
Table dropped. 0.06 seconds				

6. Rename the EMPLOYEES2 table as EMP.

QUERY:

RENAME EMPLOYEE2 to EMP;

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. Below the toolbar, a command line shows the RENAME statement: "1 RENAME EMPLOYEE2 to EMP;". The main area is titled "Results" and contains the output: "Statement processed." and "0.05 seconds". There are also tabs for Explain, Describe, Saved SQL, and History.

```
Language SQL Rows 10 Clear Command Find Tables
1 RENAME EMPLOYEE2 to EMP;
Statement processed.
0.05 seconds
```

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

QUERY:

```
COMMENT ON TABLE DEPT  
IS'DEPARTMENT_INFO'; COMMENT ON TABLE EMP  
IS 'EMPLOYEE_INFO';
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. Below the toolbar, there is a toolbar with icons for refresh, search, and other functions. The main area contains the following SQL code:

```
1 COMMENT ON TABLE DEPT IS'DEPARTMENT_INFO';  
2 COMMENT ON TABLE EMP IS 'EMPLOYEE_INFO';
```

Below the code, there is a results tab bar with options: Results (selected), Explain, Describe, Saved SQL, and History. The results section displays the message "Statement processed." and "0.03 seconds".

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

QUERY:

```
ALTER TABLE EMP DROP COLUMN First_name;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are tabs for Language (set to SQL), Rows (set to 10), and various icons for clearing the command, finding tables, and saving/run. The main area contains the SQL command: `ALTER TABLE EMP DROP COLUMN First_name;`. Below the command, the results tab is selected, showing the output: `Table altered.`. The execution time is listed as `0.07 seconds`.

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

MANIPULATING DATA

EX.NO:2

DATE: 16 - 2 - 24

REG.NO: 220701239

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)

QUERY:

```
CREATE TABLE MY_EMPLOYEE( ID number(4), Last_name varchar(25),
First_name varchar(25), Userid varchar(25), Salary number(9,2));
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. Below the toolbar, there are icons for Undo, Redo, Search, and Paste. The main area contains a command line with the following SQL code:

```
1 CREATE TABLE MY_EMPLOYEE( ID number(4), Last_name varchar(25),
First_name varchar(25), Userid varchar(25), Salary number(9,2));
```

Below the command line, there is a results pane. The tab bar at the top of the pane includes Results, Explain, Describe, Saved SQL, and History. The Results tab is selected. The output in the pane shows:

Table created.
0.03 seconds

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

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

QUERY:

```
INSERT INTO MY_EMPLOYEE VALUES(1,'Patel','Ralph','rpatel',895);
INSERT INTO MY_EMPLOYEE
VALUES(2,'Dancs','Betty','bdancs',860);
```

OUTPUT:

The screenshot shows a MySQL command-line interface. The SQL tab is selected, and the command window contains the following SQL code:

```
1 INSERT INTO MY_EMPLOYEE VALUES(1,'Patel','Ralph','rpatel',895);
2 INSERT INTO MY_EMPLOYEE VALUES(2,'Dancs','Betty','bdancs',860);
```

The results pane at the bottom shows the output:

```
1 row(s) inserted.
0.00 seconds
```

3. Display the table with values.

QUERY:

```
SELECT * FROM MY_EMPLOYEE
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. At the top, there are tabs for APEX, App Builder, SQL Workshop (which is selected), Team Development, and Gallery. On the right side, there's a user profile for "Vignesh C" (rec_317) and a schema dropdown set to "WKSP_ANANDSANTHOSH". The main area has a search bar and a toolbar with various icons. Below the toolbar, the SQL command "SELECT * FROM MY_EMPLOYEE" is entered in the SQL Commands section. The results are displayed in a table format below, showing two rows of data. The table has columns: ID, LAST_NAME, FIRST_NAME, USERID, and SALARY. The data is as follows:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860

Below the table, it says "2 rows returned in 0.03 seconds" and there are download options. At the bottom, there are footer links for user info, copyright notice (Copyright © 1999, 2023, Oracle and/or its affiliates.), and the version "Oracle APEX 23.2.4".

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

QUERY:

```
INSERT INTO MY_EMPLOYEE VALUES(3,'Biri','Ben','bbiri',1100);
INSERT INTO MY_EMPLOYEE VALUES(4,'Newman','Chad','cnewman',750);
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. Below the toolbar, the SQL command is displayed:

```
1 INSERT INTO MY_EMPLOYEE VALUES(3,'Biri','Ben','bbiri',1100);
2 INSERT INTO MY_EMPLOYEE VALUES(4,'Newman','Chad','cnewman',750);
```

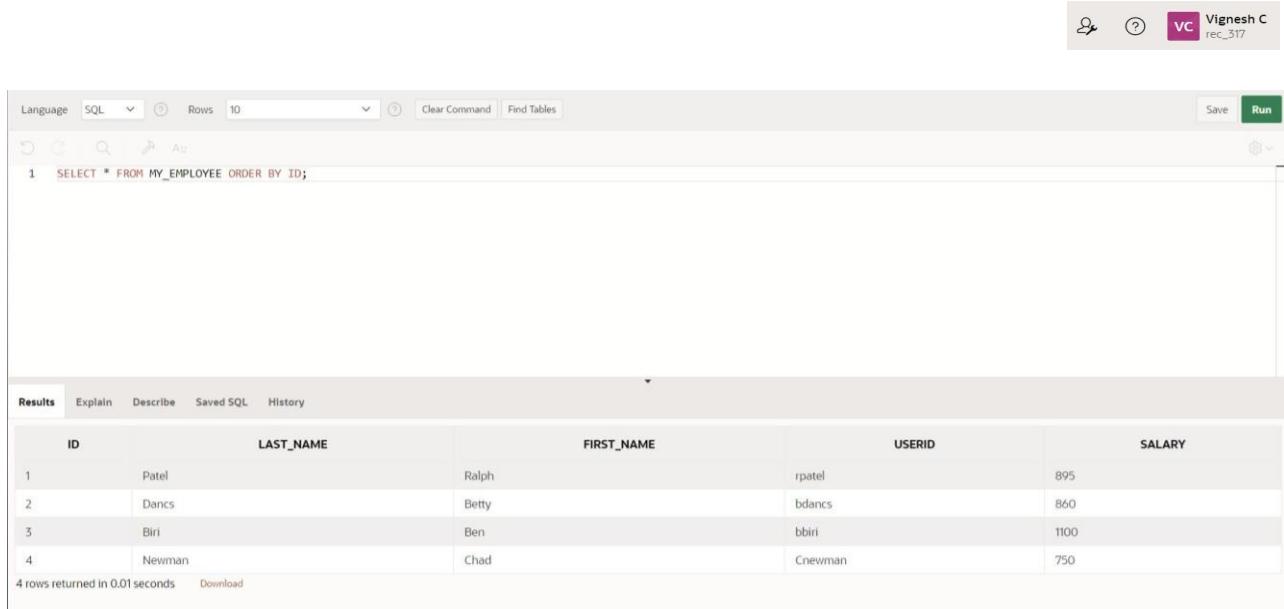
Below the command area, there is a results tab bar with tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected. The output section displays the message "1 row(s) inserted." and "0.00 seconds".

5. Make the data additions permanent.

QUERY:

```
SELECT * FROM MY_EMPLOYEE ORDER BY ID;
```

OUTPUT:



The screenshot shows a SQL query execution interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. A user profile icon for "Vignesh C" (rec_317) is also present. The query entered is "SELECT * FROM MY_EMPLOYEE ORDER BY ID;". The results section displays a table with four rows of employee 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

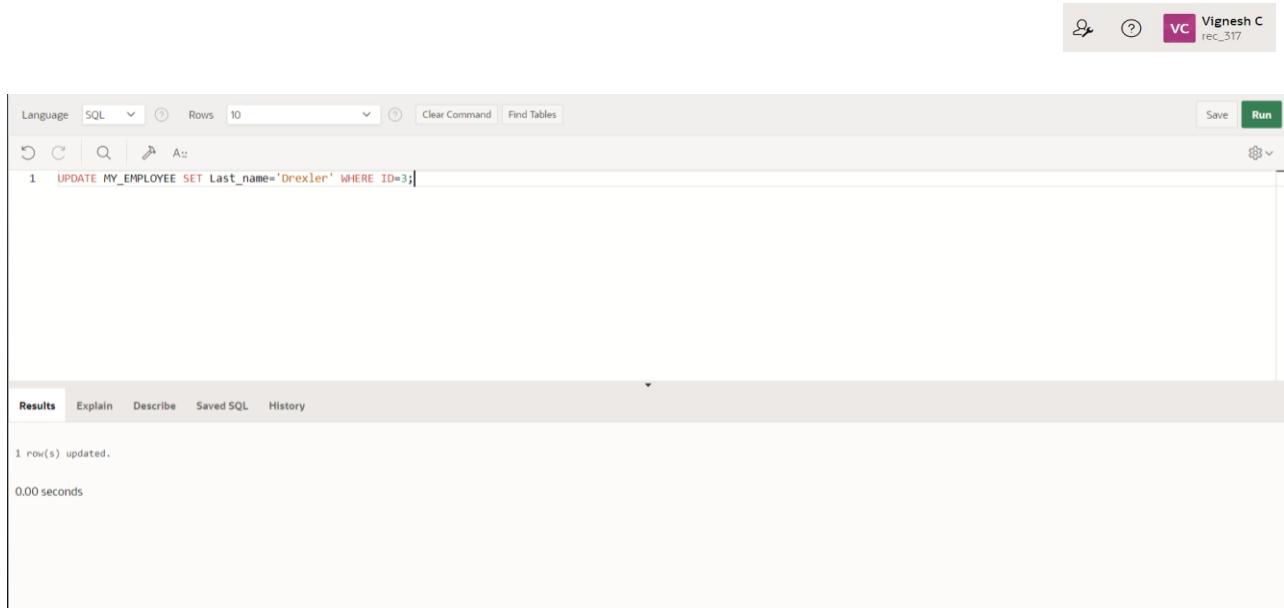
Below the table, it says "4 rows returned in 0.01seconds" and has a "Download" link.

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

QUERY:

```
UPDATE MY_EMPLOYEE SET Last_name='Drexler' WHERE ID=3;
```

OUTPUT:



The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. The Run button is highlighted in green. In the main area, a single line of SQL code is entered: `1 UPDATE MY_EMPLOYEE SET Last_name='Drexler' WHERE ID=3;`. Below the code, the results tab is selected, showing the output: `1 row(s) updated.` and `0.00 seconds`.

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

QUERY:

```
UPDATE MY_EMPLOYEE SET Salary=1000 WHERE Salary<900;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are tabs for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. A user profile icon for 'Vignesh C' is also visible. The main area contains the SQL command:

```
1 UPDATE MY_EMPLOYEE SET Salary=1000 WHERE Salary<900;
```

Below the command, there is a results section with tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected. The output shows:

3 row(s) updated.
0.00 seconds

8. Delete Betty dancs from MY_EMPLOYEE table.

QUERY:

```
DELETE FROM MY_EMPLOYEE WHERE First_name='Betty';
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are tabs for Language (set to SQL), Rows (set to 10), and buttons for Clear Command and Find Tables. On the right, there are Save and Run buttons, and a user profile icon for 'Vignesh C' with the identifier 'rec_317'. The main area contains the SQL command:

```
1  DELETE FROM MY_EMPLOYEE WHERE First_name='Betty';
```

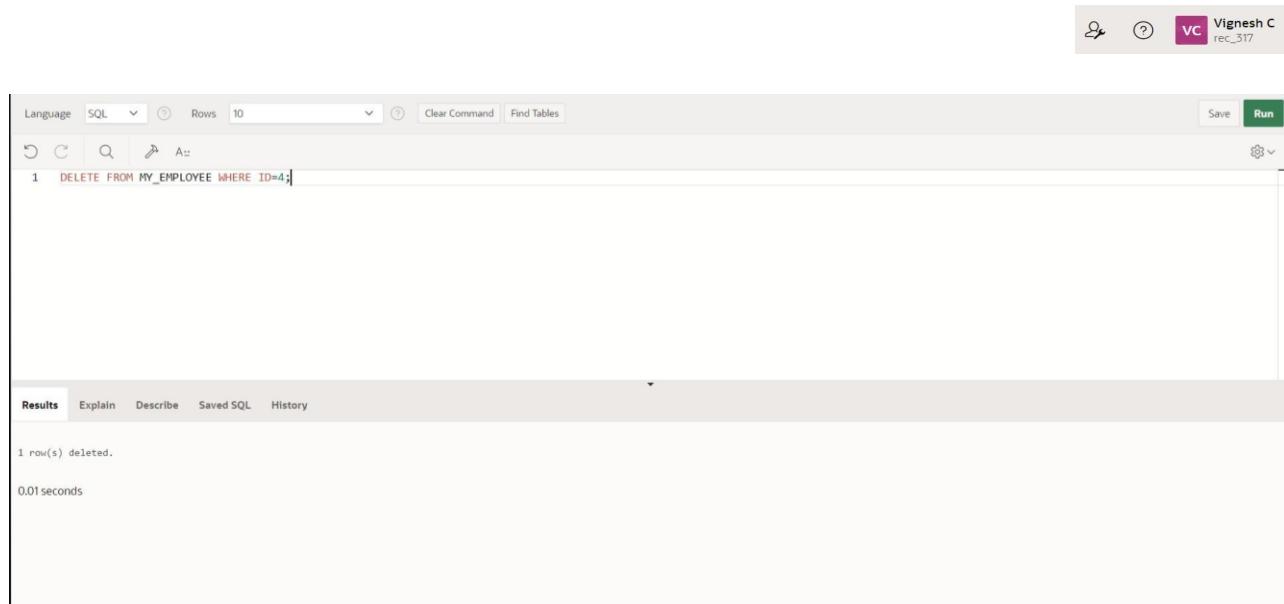
Below the command, the results tab is selected. It displays the message "1 row(s) deleted." and "0.01 seconds".

9. Empty the fourth row of the emp table.

QUERY:

```
DELETE FROM MY_EMPLOYEE WHERE ID=4;
```

OUTPUT:



The screenshot shows a MySQL command-line interface. At the top, there's a toolbar with icons for user, help, and save, followed by the session name "Vignesh C" and "rec_317". Below the toolbar, the SQL tab is selected, and the command "DELETE FROM MY_EMPLOYEE WHERE ID=4;" is entered in the command line. The results tab is active, displaying the output of the query: "1 row(s) deleted." and "0.01 seconds".

```
Language: SQL | Rows: 10 | Clear Command | Find Tables | Save | Run | User: Vignesh C | Session: rec_317 | Help |
```

```
1  DELETE FROM MY_EMPLOYEE WHERE ID=4;
```

Results	Explain	Describe	Saved SQL	History
1 row(s) deleted. 0.01 seconds				

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

INCLUDING CONSTRAINTS

EX.NO:3

DATE: 22 – 2 - 24

REG.NO: 220701239

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.

QUERY:

```
CREATE TABLE EMPLOYEES (Employ_id number(6) not null, First_name  
varchar(40), Last_name varchar(40), CONSTRAINT my_emp_id_pk PRIMARY  
KEY(Employ_id));
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. In the top navigation bar, 'APEX' is selected. The main area is titled 'SQL Commands'. The schema dropdown shows 'WKSP_ANANDSANTHOSH'. The SQL command entered is:

```
1 CREATE TABLE EMPLOYEES (Employ_id number(6) not null, First_name  
varchar(40), Last_name varchar(40), CONSTRAINT my_emp_id_pk PRIMARY KEY(Employ_id));
```

In the results tab, the output is displayed as:

```
Table created.  
0.05 seconds
```

At the bottom, the footer includes user information: '220701026@rajalakshmi.edu.in', 'anand_santhosh', and 'en'. It also states 'Copyright © 1999, 2023, Oracle and/or its affiliates.' and 'Oracle APEX 23.2.4'.

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

QUERY:

```
CREATE TABLE DEPARTMENT (Dept_id number(6) not null, First_name varchar(40),  
Last_name varchar(40), age int, CONSTRAINT my_dept_id_pk PRIMARY KEY  
(Dept_id));
```

OUTPUT:

The screenshot shows a SQL command window with the following details:

- Toolbar:** Includes Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run buttons.
- Query Editor:** Contains the SQL command: `CREATE TABLE DEPARTMENT (Dept_id number(6) not null, First_name varchar(40), Last_name varchar(40), age int, CONSTRAINT my_dept_id_pk PRIMARY KEY (Dept_id));`.
- Results Tab:** Active tab, showing the output of the query.
- Output:** Displays "Table created." and "0.05 seconds".

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

QUERY:

```
ALTER TABLE EMPLOYEES ADD(Dept_id int);
ALTER TABLE EMPLOYEES ADD CONSTRAINT my_emp_dept_id_fk FOREIGN KEY
(Dept_id) REFERENCES DEPARTMENT(Dept_id);
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. Below the toolbar, there is a text area containing the following SQL code:

```
1 ALTER TABLE EMPLOYEES ADD(Dept_id int);
2 ALTER TABLE EMPLOYEES ADD CONSTRAINT my_emp_dept_id_fk FOREIGN KEY (Dept_id) REFERENCES DEPARTMENT(Dept_id);
```

Below the code, there is a results pane with tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected. It displays the output of the executed queries:

Table altered.
0.06 seconds

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

QUERY:

```
ALTER TABLE EMPLOYEES ADD(Commision number(8));
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. The main area contains the SQL command: `ALTER TABLE EMPLOYEES ADD(Commision number(8));`. Below the command, the results are displayed: "Table altered." and "0.07 seconds". At the bottom, there are tabs for Results, Explain, Describe, Saved SQL, and History, with the Results tab selected.

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

WRITING BASIC SQL SELECT STATEMENTS

EX.NO:4

DATE: 29 - 2 - 24

REG.NO: 220701239

1. The following statement executes successfully.

Identify the Errors

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

QUERY:

```
SELECT EMPLOYEE_ID,Last_name ,Salary*12 as Annual salary  
From employee;
```

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

QUERY:

```
DESC DEPARTMENT;
```

OUTPUT:

The screenshot shows the MySQL Workbench interface with the following details:

- Language: SQL
- Rows: 10
- Clear Command, Find Tables buttons
- Run button
- SQL command: `1 DESC DEPARTMENT;`
- Results tab selected
- Object Type: TABLE
- Object: DEPARTMENT
- Table structure:

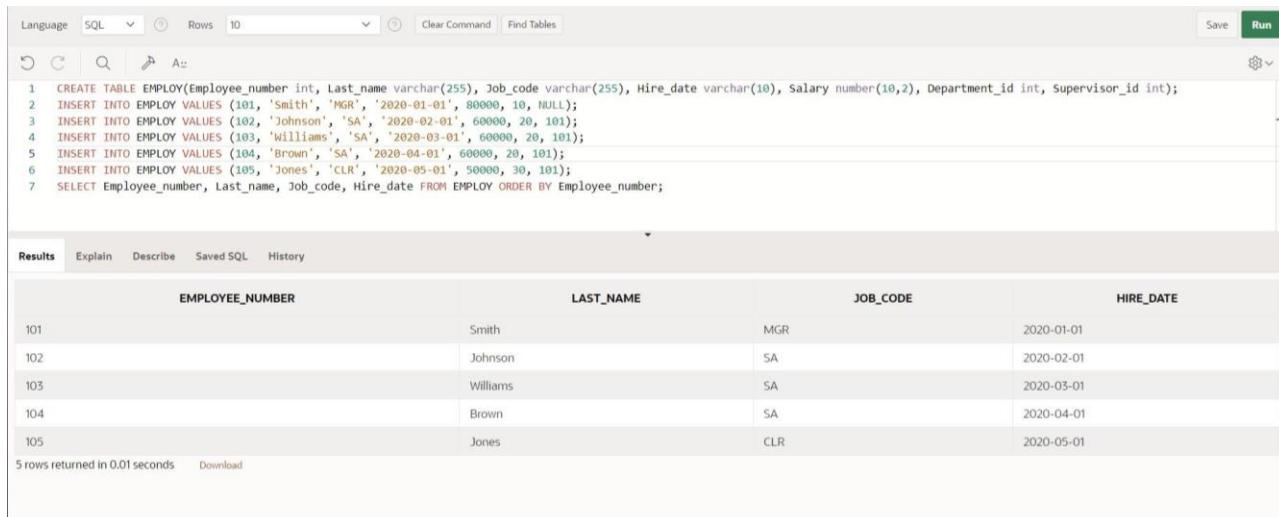
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
DEPARTMENT	DEPT_ID	NUMBER	-	6	0	1	-	-	-
	FIRST_NAME	VARCHAR2	40	-	-	-	✓	-	-
	LAST_NAME	VARCHAR2	40	-	-	-	✓	-	-
	AGE	NUMBER	22	-	0	-	✓	-	-

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

QUERY:

```
CREATE TABLE EMPLOY(Employee_number int, Last_name varchar(255), Job_code  
varchar(255), Hire_date varchar(10), Salary number(10,2), Department_id int, Supervisor_id int);  
  
INSERT INTO EMPLOY VALUES (101, 'Smith', 'MGR', '2020-01-01', 80000, 10, NULL);  
  
INSERT INTO EMPLOY VALUES (102, 'Johnson', 'SA', '2020-02-01', 60000, 20, 101);  
  
INSERT INTO EMPLOY VALUES (103, 'Williams', 'SA', '2020-03-01', 60000, 20, 101);  
  
INSERT INTO EMPLOY VALUES (104, 'Brown', 'SA', '2020-04-01', 60000, 20, 101);  
  
INSERT INTO EMPLOY VALUES (105, 'Jones', 'CLR', '2020-05-01', 50000, 30, 101);  
  
SELECT Employee_number, Last_name, Job_code, Hire_date FROM EMPLOY ORDER  
BY Employee_number;
```

OUTPUT:



The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. Below the toolbar, the SQL script is displayed:

```
1 CREATE TABLE EMPLOY(Employee_number int, Last_name varchar(255), Job_code varchar(255), Hire_date varchar(10), Salary number(10,2), Department_id int, supervisor_id int);  
2 INSERT INTO EMPLOY VALUES (101, 'Smith', 'MGR', '2020-01-01', 80000, 10, NULL);  
3 INSERT INTO EMPLOY VALUES (102, 'Johnson', 'SA', '2020-02-01', 60000, 20, 101);  
4 INSERT INTO EMPLOY VALUES (103, 'Williams', 'SA', '2020-03-01', 60000, 20, 101);  
5 INSERT INTO EMPLOY VALUES (104, 'Brown', 'SA', '2020-04-01', 60000, 20, 101);  
6 INSERT INTO EMPLOY VALUES (105, 'Jones', 'CLR', '2020-05-01', 50000, 30, 101);  
7 SELECT Employee_number, Last_name, Job_code, Hire_date FROM EMPLOY ORDER BY Employee_number;
```

The Results tab is selected, showing the following table output:

EMPLOYEE_NUMBER	LAST_NAME	JOB_CODE	HIRE_DATE
101	Smith	MGR	2020-01-01
102	Johnson	SA	2020-02-01
103	Williams	SA	2020-03-01
104	Brown	SA	2020-04-01
105	Jones	CLR	2020-05-01

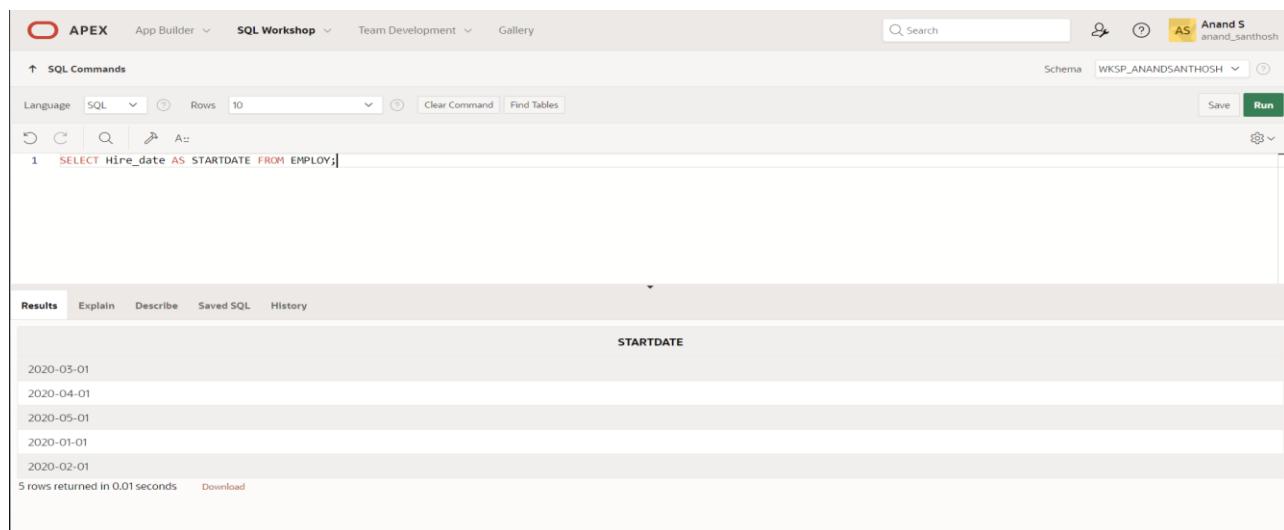
At the bottom left, it says "5 rows returned in 0.01 seconds".

4. Provide an alias STARTDATE for the hire date.

QUERY:

```
SELECT Hire_date AS STARTDATE FROM EMPLOY;
```

OUTPUT:



The screenshot shows the Oracle SQL Workshop interface. In the SQL Commands pane, the query `SELECT Hire_date AS STARTDATE FROM EMPLOY;` is entered. The Results pane displays the output:

STARTDATE
2020-03-01
2020-04-01
2020-05-01
2020-01-01
2020-02-01

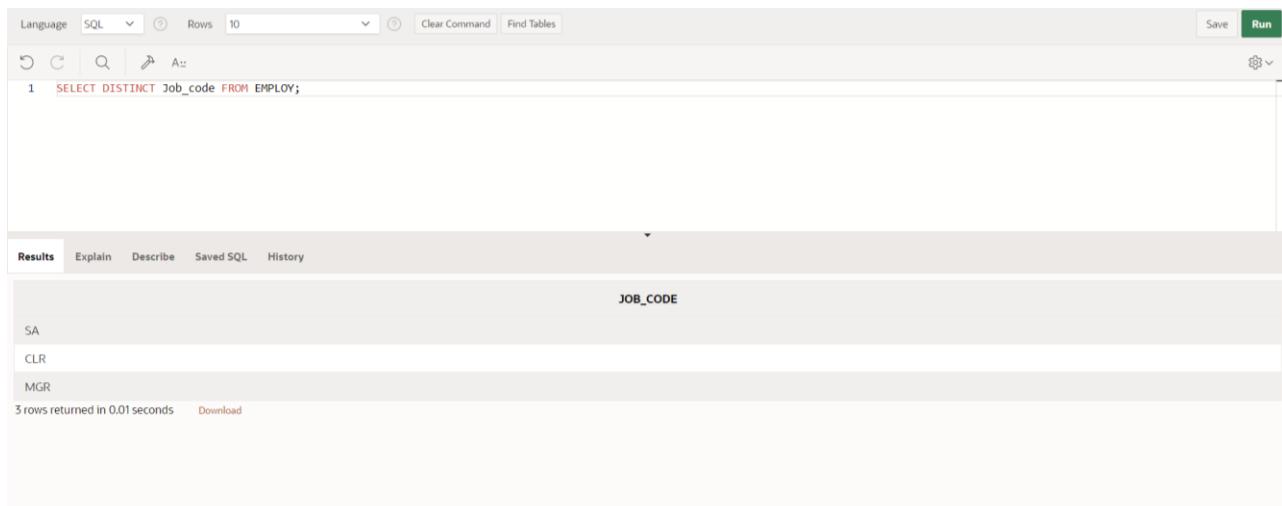
5 rows returned in 0.01 seconds

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

QUERY:

```
SELECT DISTINCT Job_code FROM EMPLOY;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

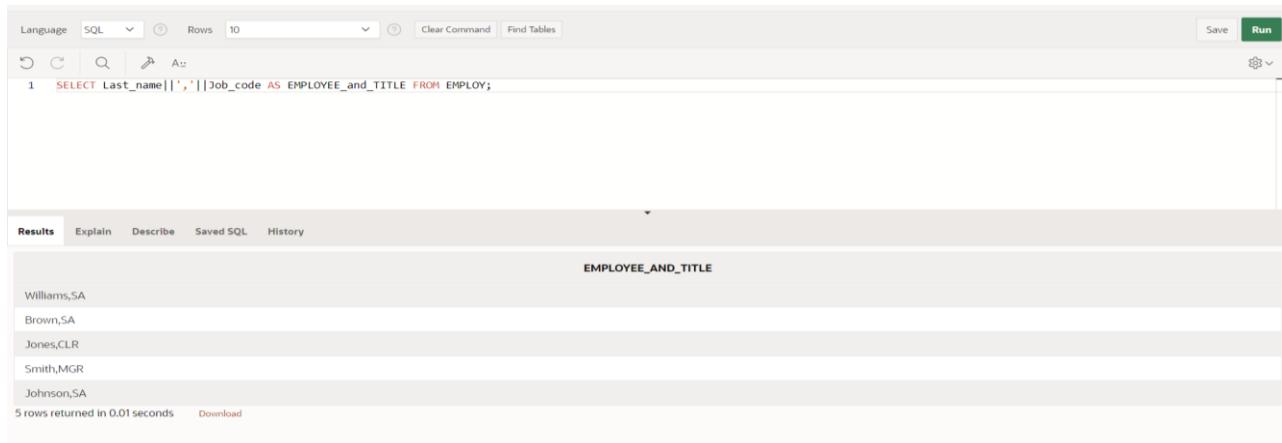
- Language: SQL
- Rows: 10
- Clear Command
- Find Tables
- Run button
- Query entered: `SELECT DISTINCT Job_code FROM EMPLOY;`
- Results tab selected
- Table header: `JOB_CODE`
- Data rows:
 - SA
 - CLR
 - MGR
- Message: 3 rows returned in 0.01 seconds
- Download link

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

QUERY:

```
SELECT Last_name||','||Job_code AS EMPLOYEE_and_TITLE FROM EMPLOY;
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
1 SELECT Last_name||','||Job_code AS EMPLOYEE_and_TITLE FROM EMPLOY;
```

The results section displays the output:

EMPLOYEE_AND_TITLE
Williams,SA
Brown,SA
Jones,CLR
Smith,MGR
Johnson,SA

5 rows returned in 0.01 seconds [Download](#)

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

QUERY:

```
SELECT Last_name||','||Job_code||','||Employee_number||','||Hire_date AS THE_OUTPUT FROM EMPLOY;
```

OUTPUT:

The screenshot shows a SQL query execution interface. The query is:

```
1 SELECT Last_name||','||Job_code||','||Employee_number||','||Hire_date AS THE_OUTPUT FROM EMPLOY;
```

The results are displayed in a table with one column labeled "THE_OUTPUT". The data consists of five rows:

THE_OUTPUT
Williams,SA,103,2020-03-01
Brown,SA,104,2020-04-01
Jones,CLR,105,2020-05-01
Smith,MGR,101,2020-01-01
Johnson,SA,102,2020-02-01

5 rows returned in 0.01 seconds

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

RESTRICTING AND STORING DATA

EX.NO:5

DATE: 01 – 3 - 24

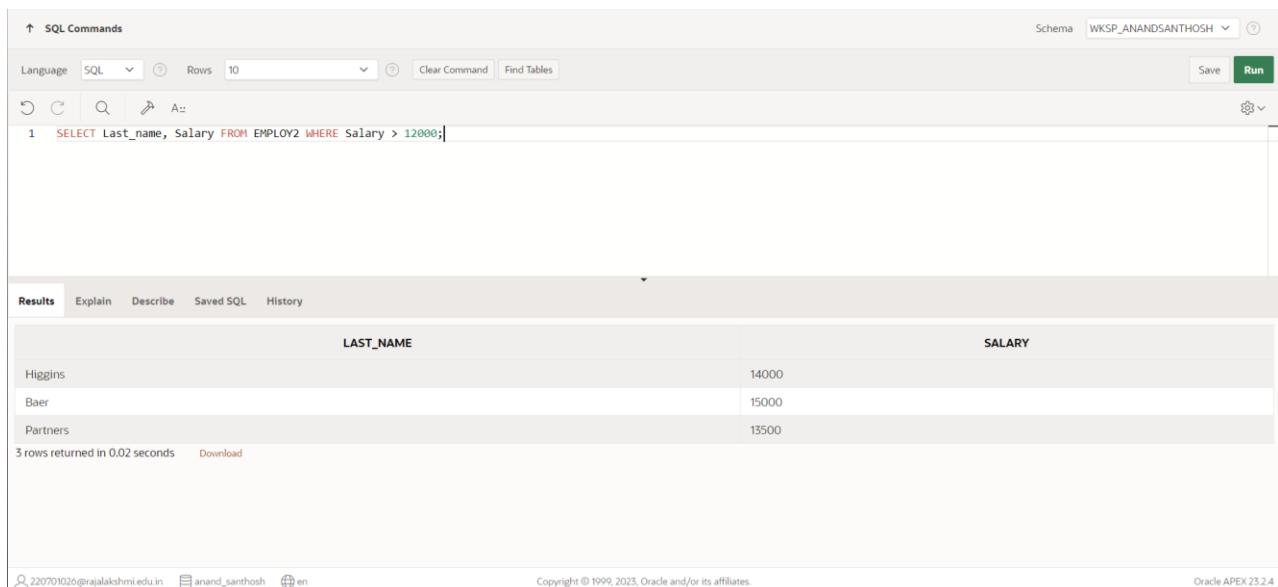
REG.NO: 220701239

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

QUERY:

```
SELECT Last_name, Salary FROM EMPLOY2 WHERE Salary > 12000;
```

OUTPUT:



The screenshot shows the Oracle APEX SQL Commands interface. The query `SELECT Last_name, Salary FROM EMPLOY2 WHERE Salary > 12000;` is entered in the command field. The results page displays the output in a table format:

LAST_NAME	SALARY
Higgins	14000
Bae	15000
Partners	13500

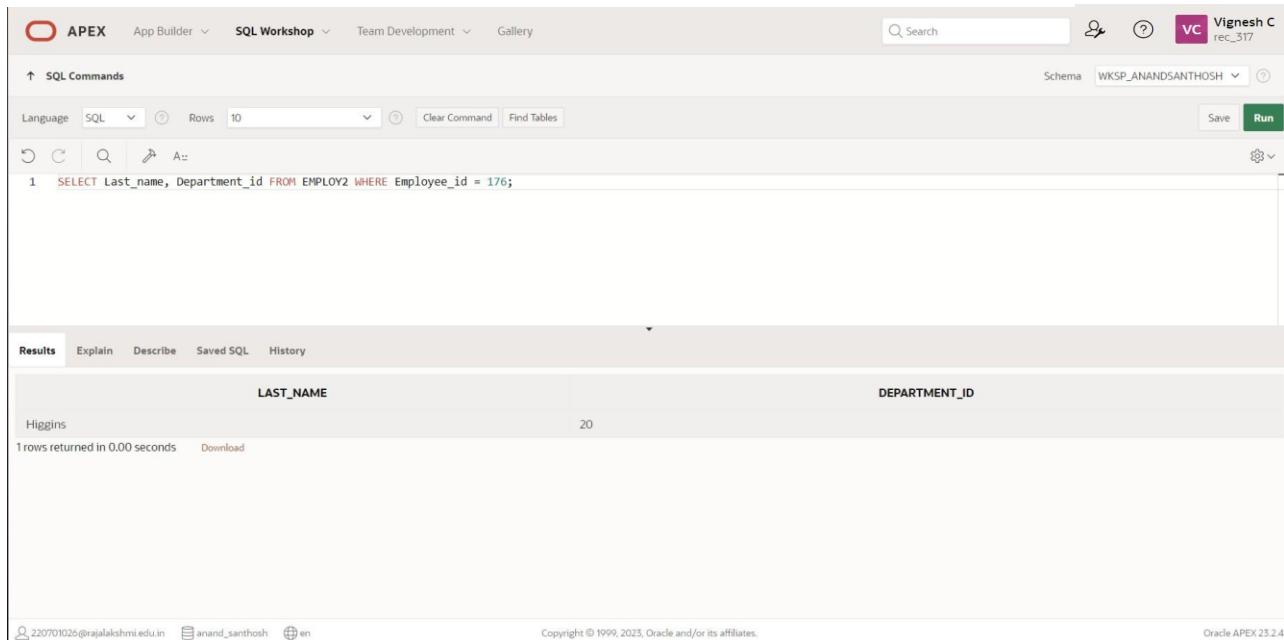
3 rows returned in 0.02 seconds. The bottom of the screen shows user information and copyright details.

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

QUERY:

```
SELECT Last_name, Department_id FROM EMPLOY2 WHERE Employee_id = 176;
```

OUTPUT:



The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. The right side shows a user profile for 'Vignesh C' (rec_317) and a schema dropdown set to 'WKSP_ANANDSANTHOSH'. The main area is titled 'SQL Commands' with a language dropdown set to 'SQL'. The command entered is: 'SELECT Last_name, Department_id FROM EMPLOY2 WHERE Employee_id = 176;'. Below the command, the results tab is selected, showing a single row of data: Higgins in the LAST_NAME column and 20 in the DEPARTMENT_ID column. The status bar at the bottom indicates '1 rows returned in 0.00 seconds' and provides download options. The footer includes copyright information for Oracle and the APEX version 'Oracle APEX 23.2.4'.

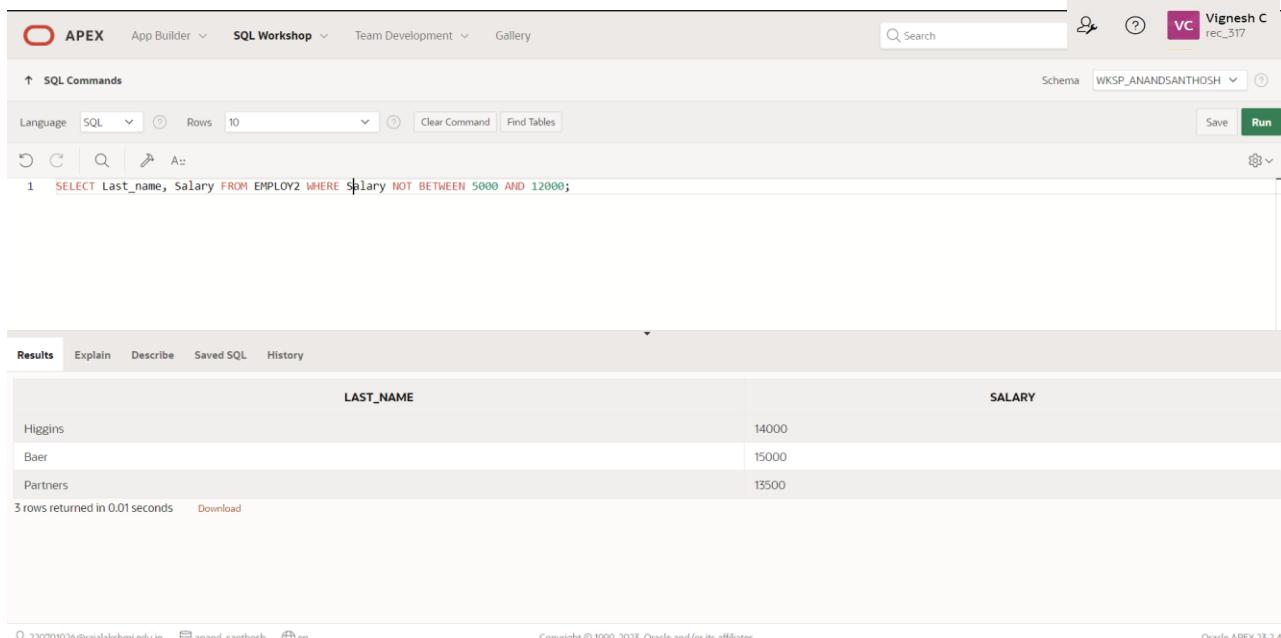
LAST_NAME	DEPARTMENT_ID
Higgins	20

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

QUERY:

```
SELECT Last_name, Salary FROM EMPLOY2 WHERE Salary NOT BETWEEN 5000 AND 12000;
```

OUTPUT:



The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. On the right, there's a user profile for 'Vignesh C' (rec_317) with a search bar and schema dropdown set to 'WKSP_ANANDSANTHOSH'. The main workspace displays the SQL command entered:

```
1 SELECT Last_name, Salary FROM EMPLOY2 WHERE Salary NOT BETWEEN 5000 AND 12000;
```

Below the command, the results tab is selected, showing the output of the query:

LAST_NAME	SALARY
Higgins	14000
Baer	15000
Partners	13500

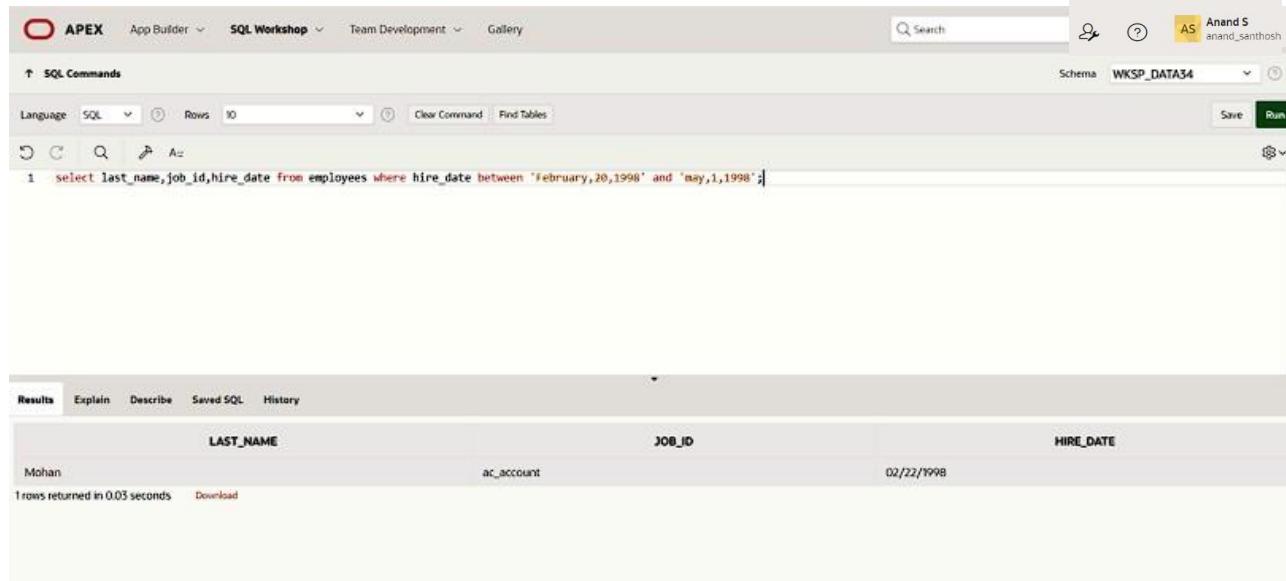
3 rows returned in 0.01 seconds [Download](#)

At the bottom, footer information includes the URL 220701026@rajalakshmi.edu.in, the user name anand_santhosh, and the language en. It also states Copyright © 1999, 2023, Oracle and/or its affiliates, and Oracle APEX 23.2.4.

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

QUERY:

OUTPUT:



The screenshot shows the Oracle SQL Workshop interface. The top navigation bar includes APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. On the right, it shows the user 'Anand S' and schema 'WKSP_DATA34'. The main area is titled 'SQL Commands' with a 'Run' button. The command entered is:

```
1 select last_name,job_id,hire_date from employees where hire_date between 'February,20,1998' and 'may,1,1998';
```

The results tab is selected, displaying the following data:

LAST_NAME	JOB_ID	HIRE_DATE
Mohan	ac_account	02/22/1998

Below the table, it says '1 rows returned in 0.03 seconds'.

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

QUERY:

```
SELECT Last_name, Department_id FROM EMPLOY2 WHERE Department_id IN (20, 50)  
ORDER BY Last_name;
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. The right side of the header shows a user profile for 'Vignesh C' (rec_317) and a schema dropdown set to 'WKSP_ANANDSANTHOSH'. The main workspace has tabs for SQL Commands, Results, Explain, Describe, Saved SQL, and History. The SQL Commands tab contains the executed query: 'SELECT Last_name, Department_id FROM EMPLOY2 WHERE Department_id IN (20, 50) ORDER BY Last_name;'. The Results tab displays the output in a table format:

LAST_NAME	DEPARTMENT_ID
Baer	50
De Haan	20
Ernst	20
Gietz	20
Hartstein	20
Higgins	20
Hunold	50
Kochhar	20
Lorentz	50
Sciarra	50

Below the table, a message states '10 rows returned in 0.02 seconds' and includes a 'Download' link. The bottom of the page shows user information (220701025@rajalakshmi.edu.in, anand_santhosh, en), a copyright notice (Copyright © 1999, 2023, Oracle and/or its affiliates.), and the software version (Oracle APEX 23.2.4).

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

QUERY:

```
SELECT Last_name, Job_id, Hire_date FROM EMPLOY2 WHERE Salary BETWEEN 5000  
AND 12000 AND Department_id IN (20, 50) ORDER BY Last_name;
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. At the top, there are tabs for APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. On the right, there's a search bar, a user icon for 'Vignesh C rec_317', and a schema dropdown set to 'WKSP_ANANDSANTHOSH'. The main area has a toolbar with icons for Undo, Redo, Find, Copy, Paste, and Run. Below the toolbar, the SQL command is displayed:

```
1 SELECT Last_name, Job_id, Hire_date FROM EMPLOY2 WHERE Salary BETWEEN 5000 AND 12000 AND Department_id IN (20, 50) ORDER BY Last_name;
```

Below the command, there are tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected, showing a table with three columns: LAST_NAME, JOB_ID, and HIRE_DATE. The data returned is:

LAST_NAME	JOB_ID	HIRE_DATE
Ernst	SA_REP	1991-07-01
Gietz	SA_REP	1994-05-17
Hartstein	SA_REP	1996-05-01
Hunold	ST_CLERK	1990-08-05
Lorentz	ST_CLERK	1997-04-01
Sciarrra	ST_CLERK	1993-06-14

At the bottom left, it says '6 rows returned in 0.00 seconds' and 'Download'. At the bottom right, it says 'Copyright © 1999, 2023, Oracle and/or its affiliates.' and 'Oracle APEX 23.2.4'.

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

QUERY:

```
SELECT Last_name, Hire_date FROM EMPLOY2 WHERE Hire_date LIKE '1994-%-%';
```

OUTPUT:

The screenshot shows the Oracle APEX interface with the SQL Workshop tab selected. The query `SELECT Last_name, Hire_date FROM EMPLOY2 WHERE Hire_date LIKE '1994-%-%';` is entered in the command line. The results section displays one row: Gietz, hired on 1994-05-17.

LAST_NAME	HIRE_DATE
Gietz	1994-05-17

1 rows returned in 0.00 seconds Download

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

QUERY:

```
SELECT Last_name, Job_id FROM EMPLOY2 WHERE Manager_id IS NULL;
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop, Team Development, and Gallery. On the right side, there is a user profile for "Vignesh C" and a schema dropdown set to "WKSP_ANANDSANTHOSH". The main workspace has a toolbar with various icons. Below the toolbar, the SQL command input field contains the query: "SELECT Last_name, Job_id FROM EMPLOY2 WHERE Manager_id IS NULL;". The results tab is selected, displaying the output in a grid format:

LAST_NAME	JOB_ID
Higgins	SA_MAN
Baer	ST_MAN
De Haan	SA_MAN
Kochhar	SA_MAN

Below the grid, it says "4 rows returned in 0.01 seconds" and there is a "Download" link. At the bottom of the page, there are footer links for "22070102@rajalakshmi.edu.in", "anand_santhosh", and "en", along with copyright information: "Copyright © 1999, 2023, Oracle and/or its affiliates." and "Oracle APEX 25.2.4".

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.(hints: is not null,order by)

QUERY:

```
SELECT Last_name, Salary, Commission_pct FROM EMPLOY2 WHERE Commission_pct  
IS NOT NULL ORDER BY Salary DESC, Commission_pct DESC;
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop (selected), Team Development, and Gallery. The right side of the header shows a user profile for 'Vignesh C' (rec_317) and a schema dropdown set to 'WKSP_ANANDSANTHOSH'. The main workspace has a toolbar with various icons. The SQL command input field contains the query: `SELECT Last_name, Salary, Commission_pct FROM EMPLOY2 WHERE Commission_pct IS NOT NULL ORDER BY Salary DESC, Commission_pct DESC;`. Below the command is a results grid. The grid has three columns: LAST_NAME, SALARY, and COMMISSION_PCT. The data is as follows:

LAST_NAME	SALARY	COMMISSION_PCT
Kochhar	17000	0
Baer	15000	0
Higgins	14000	0
De Haan	14000	0
Partners	13500	0
Hartstein	12000	0
Fay	11000	0
Greenberg	11000	0
Lorentz	9000	0
Hunold	9000	0

Below the grid, a message says 'More than 10 rows available. Increase rows selector to view more rows.' At the bottom, it shows '10 rows returned in 0.01 seconds' and provides download options. The footer includes copyright information for Oracle and the APEX version.

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

QUERY:

```
SELECT Last_name FROM EMPLOY2 WHERE Last_name LIKE '_a%'
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop' (selected), 'Team Development', and 'Gallery'. The right side shows a user profile for 'Vignesh C' (rec_317). The main area has tabs for 'SQL Commands' and 'Results'. In the 'SQL Commands' tab, the query 'SELECT Last_name FROM EMPLOY2 WHERE Last_name LIKE '_a%' is entered. The 'Results' tab displays the output in a table with a single column 'LAST_NAME'. The results show two rows: 'Faahadh' and 'Scarra'. At the bottom, it says '2 rows returned in 0.00 seconds' and provides a 'Download' link.

LAST_NAME
Faahadh
Scarra

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

QUERY:

```
SELECT Last_name FROM EMPLOY2 WHERE Last_name LIKE '%a%' AND Last_name LIKE '%e%';
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes 'APEX', 'App Builder', 'SQL Workshop' (selected), 'Team Development', and 'Gallery'. On the right, there's a user profile for 'Vignesh C' (rec_317) and a schema dropdown set to 'WKSP_ANANDSANHOSH'. The main workspace has tabs for 'SQL Commands' and 'Results'. Under 'SQL Commands', the query is displayed:

```
1 SELECT Last_name FROM EMPLOY2 WHERE Last_name LIKE '%a%' AND Last_name LIKE '%e%';
```

Under the 'Results' tab, the output is shown in a table with a single column 'LAST_NAME'. The results are:

LAST_NAME
Baer
Hartstein
De Haan
Partners

Below the table, it says '4 rows returned in 0.00 seconds' and has a 'Download' link. At the bottom of the page, there are footer links for email, social media, and language selection, along with copyright and version information.

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

QUERY:

```
SELECT Last_name, Job_ID, Salary FROM EMPLOY2 WHERE (Job_ID = 'SA_REP' OR Job_ID = 'ST_CLERK') AND Salary NOT IN (2500, 3500, 7000);
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop, Team Development, and Gallery. On the right side, there are user profile icons for 'Vignesh C' and 'rec_317'. The main workspace has tabs for 'SQL Commands' and 'Results'. The 'SQL Commands' tab contains the executed SQL query. The 'Results' tab displays the output in a grid format with columns: LAST_NAME, JOB_ID, and SALARY. The results show nine rows of employee data, including Hunold, Hartstein, Fay, Greenberg, Partners, Lorentz, Gietz, Ernst, and Sciarra, with their respective job IDs and salaries.

LAST_NAME	JOB_ID	SALARY
Hunold	ST_CLERK	9000
Hartstein	SA_REP	12000
Fay	SA_REP	11000
Greenberg	SA_REP	11000
Partners	SA_REP	13500
Lorentz	ST_CLERK	9000
Gietz	SA_REP	8600
Ernst	SA_REP	6000
Sciarra	ST_CLERK	7700

13. Display the last name, salary, and commission for all employees whose commission amount is 20%. (hints: use predicate logic)

QUERY:

```
SELECT Last_name, Salary, Commission_pct FROM EMPLOY2 WHERE Commission_pct = 0.2;
```

OUTPUT:

The screenshot shows the Oracle APEX SQL Workshop interface. The top navigation bar includes links for APEX, App Builder, SQL Workshop, Team Development, and Gallery. The SQL Workshop tab is active. The schema dropdown is set to WKSP_ANANDSANTHOSH. The main area displays the SQL command:

```
1 SELECT Last_name, Salary, Commission_pct FROM EMPLOY2 WHERE Commission_pct = 0.2;
```

The results tab is selected, showing the output of the query:

LAST_NAME	SALARY	COMMISSION_PCT
Higgins	14000	.2
Greenberg	11000	.2
Lorentz	9000	.2
Sciarra	7700	.2

Below the table, it says "4 rows returned in 0.01 seconds". The bottom of the page includes user information (220701025@rajalakshmi.edu.in, anand_santhosh, en), a copyright notice (Copyright © 1999, 2023, Oracle and/or its affiliates.), and a footer note (Oracle APEX 23.2.4).

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

SINGLE ROW FUNCTIONS

EX.NO:6

DATE: 07 – 3 - 24

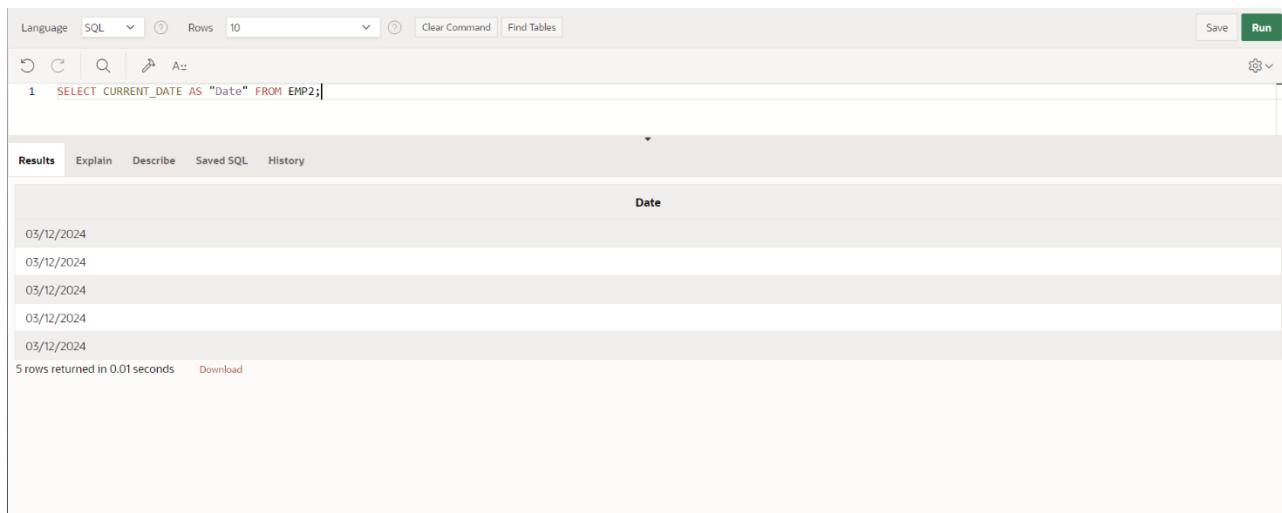
REG.NO: 220701239

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

QUERY:

```
SELECT CURRENT_DATE AS "Date" FROM EMP2;
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
1 SELECT CURRENT_DATE AS "Date" FROM EMP2;
```

The results section displays the output:

Date
03/12/2024
03/12/2024
03/12/2024
03/12/2024
03/12/2024

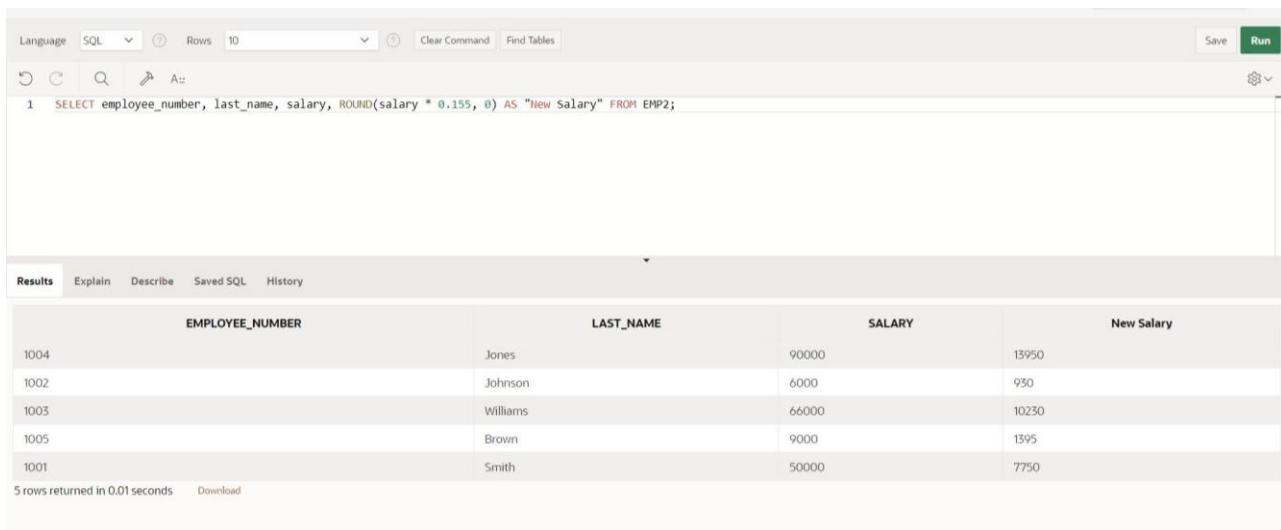
Below the table, it says "5 rows returned in 0.01 seconds".

2. The HR department needs a report to display the employee number, last name, salary, and increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary.

QUERY:

```
SELECT employee_number, last_name, salary, ROUND(salary * 0.155, 0) AS "New Salary" FROM EMP2;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- Clear Command | Find Tables | Save | Run
- Query: `SELECT employee_number, last_name, salary, ROUND(salary * 0.155, 0) AS "New Salary" FROM EMP2;`
- Results tab selected
- Table Headers: EMPLOYEE_NUMBER, LAST_NAME, SALARY, New Salary
- Data Rows:

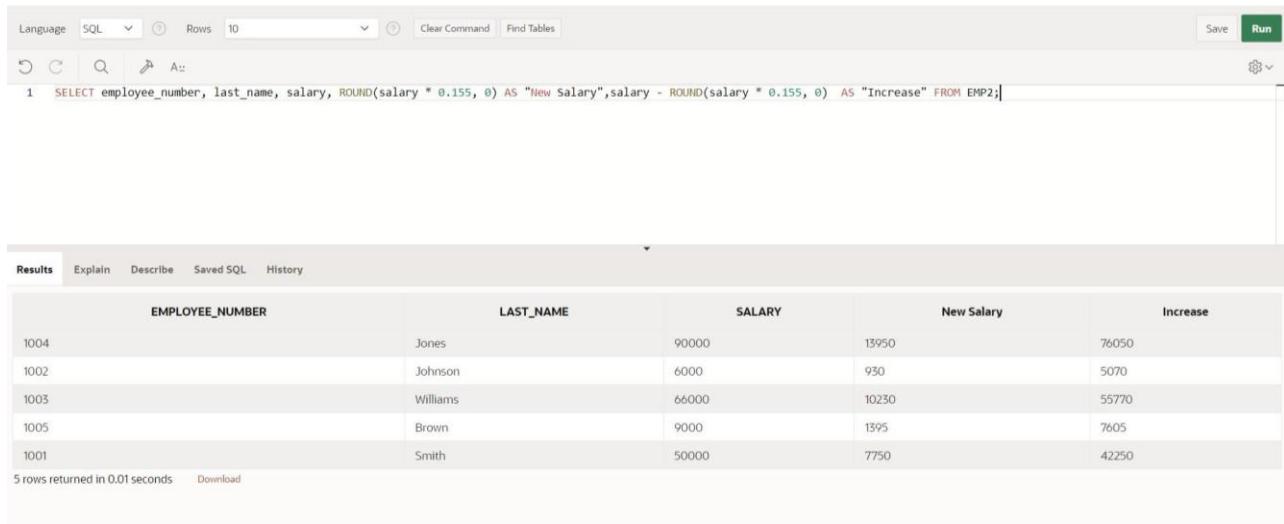
EMPLOYEE_NUMBER	LAST_NAME	SALARY	New Salary
1004	Jones	90000	13950
1002	Johnson	6000	930
1003	Williams	66000	10230
1005	Brown	9000	1395
1001	Smith	50000	7750
- Message: 5 rows returned in 0.01 seconds | Download

3. Modify your query lab_03_02.sql to add a column that subtracts the old salary from the new salary. Label the column Increase.

QUERY:

```
SELECT employee_number, last_name, salary, ROUND(salary * 0.155, 0) AS "New Salary", salary - ROUND(salary * 0.155, 0) AS "Increase" FROM EMP2;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- SQL Command:
1 SELECT employee_number, last_name, salary, ROUND(salary * 0.155, 0) AS "New Salary", salary - ROUND(salary * 0.155, 0) AS "Increase" FROM EMP2;

The Results tab is selected, displaying the query output:

EMPLOYEE_NUMBER	LAST_NAME	SALARY	New Salary	Increase
1004	Jones	90000	13950	76050
1002	Johnson	6000	930	5070
1003	Williams	66000	10230	55770
1005	Brown	9000	1395	7605
1001	Smith	50000	7750	42250

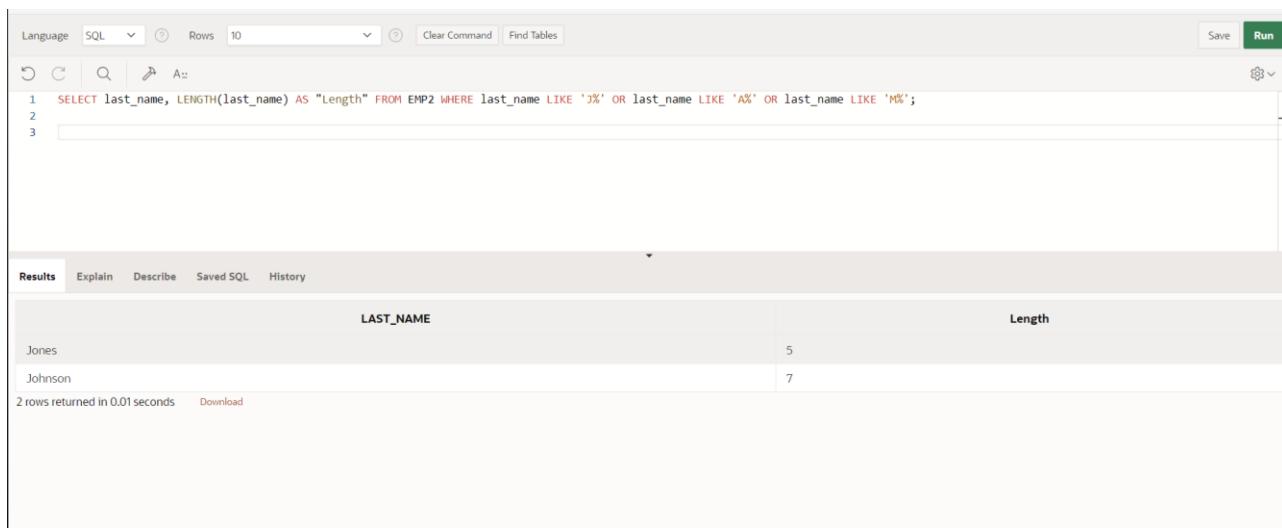
5 rows returned in 0.01 seconds Download

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

QUERY:

```
SELECT last_name, LENGTH(last_name) AS "Length" FROM EMP2 WHERE last_name LIKE 'J%' OR last_name LIKE 'A%' OR last_name LIKE 'M%';
```

OUTPUT:



The screenshot shows a SQL query editor interface with the following details:

- Language:** SQL
- Rows:** 10
- Query:**

```
1 SELECT last_name, LENGTH(last_name) AS "Length" FROM EMP2 WHERE last_name LIKE 'J%' OR last_name LIKE 'A%' OR last_name LIKE 'M%';
```
- Results:** The results table has two columns: LAST_NAME and Length. It contains two rows:

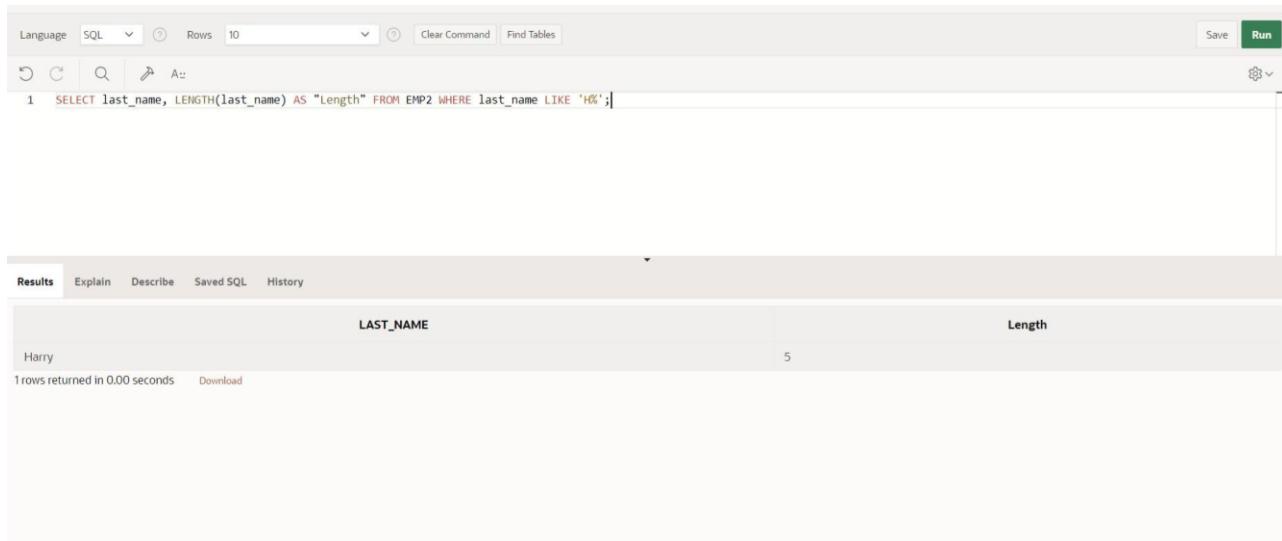
LAST_NAME	Length
Jones	5
Johnson	7
- Timing:** 2 rows returned in 0.01 seconds

5. Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, then the output should show all employees whose last name starts with the letter H.

QUERY:

```
SELECT last_name, LENGTH(last_name) AS "Length" FROM EMP2 WHERE last_name LIKE 'H%';
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
1 SELECT last_name, LENGTH(last_name) AS "Length" FROM EMP2 WHERE last_name LIKE 'H%';
```

The results section displays the following data:

LAST_NAME	Length
Harry	5

1 rows returned in 0.00 seconds Download

6. The HR department wants to find the length of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note: Your results will differ.

QUERY:

```
SELECT last_name, hire_date, ROUND(MONTHS_BETWEEN(CURRENT_DATE,  
hire_date)) AS "Months Worked" FROM EMP2 ORDER BY "Months Worked";
```

OUTPUT:

The screenshot shows a SQL query execution interface. At the top, there are tabs for Language (set to SQL), Rows (set to 10), Clear Command, and Find Tables. On the right, there are Save and Run buttons. The main area contains the SQL query:

```
1 SELECT last_name, hire_date, ROUND(MONTHS_BETWEEN(CURRENT_DATE, hire_date)) AS "Months Worked" FROM EMP2 ORDER BY "Months Worked";
```

Below the query, the Results tab is selected. The results are displayed in a table with three columns: LAST_NAME, HIRE_DATE, and Months Worked. The data is as follows:

LAST_NAME	HIRE_DATE	Months Worked
Jones	02/16/2022	25
Smith	01/01/2022	26
Williams	12/03/2021	27
Brown	09/25/2021	30
Johnson	06/15/2021	33

5 rows returned in 0.01 seconds [Download](#)

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

QUERY:

```
SELECT last_name, salary, ROUND(salary * 3, 0) AS "Dream Salaries" FROM EMP2;
```

OUTPUT:

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 10
- Clear Command | Find Tables
- Run button
- SQL command entered: `1 SELECT last_name, salary, ROUND(salary * 3, 0) AS "Dream Salaries" FROM EMP2;`
- Results tab selected
- Table output:

LAST_NAME	SALARY	Dream Salaries
Jones	90000	270000
Johnson	6000	18000
Williams	66000	198000
Brown	9000	27000
Smith	50000	150000

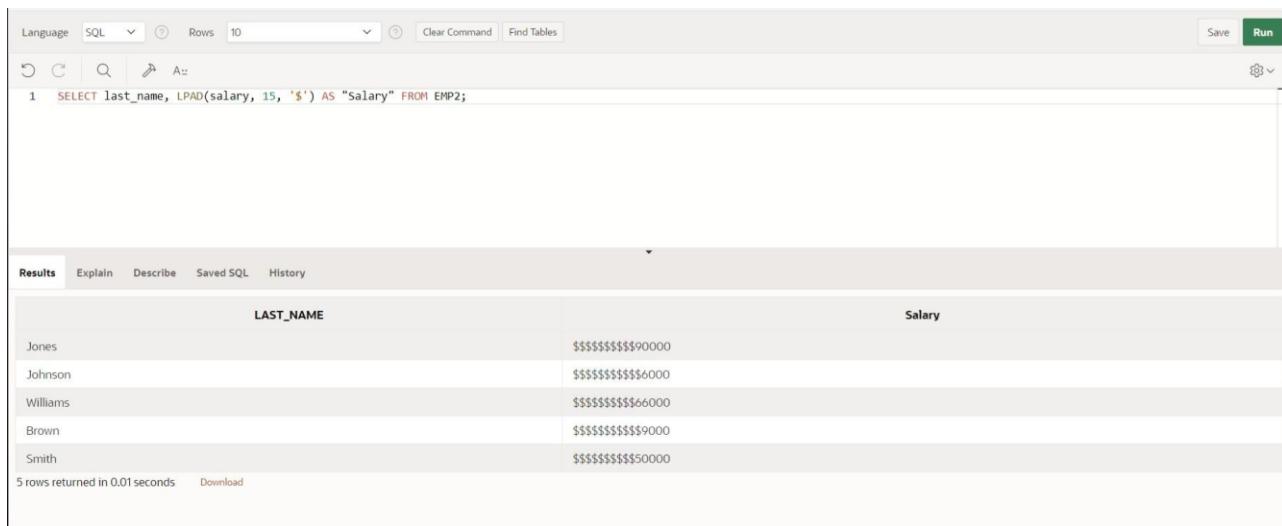
- 5 rows returned in 0.01 seconds
- Download link

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the \$ symbol. Label the column SALARY.

QUERY:

```
SELECT last_name, LPAD(salary, 15, '$') AS "Salary" FROM EMP2;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

- Language:** SQL
- Rows:** 10
- SQL Query:** `SELECT last_name, LPAD(salary, 15, '$') AS "Salary" FROM EMP2;`
- Results:** The results table has two columns: `LAST_NAME` and `Salary`. The data is as follows:

LAST_NAME	Salary
Jones	\$\$\$\$\$\$\$\$\$\$90000
Johnson	\$\$\$\$\$\$\$\$\$\$60000
Williams	\$\$\$\$\$\$\$\$\$\$66000
Brown	\$\$\$\$\$\$\$\$\$\$9000
Smith	\$\$\$\$\$\$\$\$\$\$50000

- Timing:** 5 rows returned in 0.01 seconds
- Actions:** Save, Run, Explain, Describe, Saved SQL, History, Download

9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

QUERY:

```
SELECT last_name,hire_date,TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY'),'FMDay,  
"the "FMDD "of "FMMonth, YYYY') AS REVIEW FROM employees;
```

OUTPUT:

The screenshot shows a SQL query editor interface. The top section contains the SQL code:

```
1 SELECT last_name,hire_date,TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY'),'FMDay,  
2 "the "FMDD "of "FMMonth, YYYY') AS REVIEW FROM employees;
```

The bottom section displays the results in a table format:

LAST_NAME	HIRE_DATE	REVIEW
popp	03/05/2024	Monday, the 09 of September, 2024
raphealy	02/03/1999	Monday, the 09 of August, 1999
Mohan	02/22/1998	Monday, the 24 of August, 1998

3 rows returned in 0.01 seconds Download

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

QUERY:

```
SELECT last_name, TO_CHAR(hire_date, 'Day') AS "Day" FROM EMP2 ORDER BY "Day";
```

OUTPUT:

The screenshot shows a SQL query execution interface. At the top, there are buttons for Language (SQL), Rows (set to 10), Clear Command, Find Tables, Save, and Run. Below the interface, the SQL query is displayed:

```
1 | SELECT last_name, TO_CHAR(hire_date, 'Day') AS "Day" FROM EMP2 ORDER BY "Day";|
```

The results section shows the output of the query:

LAST_NAME	Day
Williams	Friday
Smith	Saturday
Brown	Saturday
Johnson	Tuesday
Jones	Wednesday

Below the table, it says "5 rows returned in 0.01 seconds" and has a "Download" link.

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

DISPLAYING DATA FROM MULTIPLE TABLES

EX.NO:7

DATE: 14 – 3 - 24

REG.NO: 220701239

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

QUERY:

```
SELECT e.last_name, e.department_id, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id ORDER BY e.last_name;
```

OUTPUT:

The screenshot shows a MySQL command-line interface window. The query entered is:

```
1 SELECT e.last_name, e.department_id, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id ORDER BY e.last_name;
```

The results section displays the following data:

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
De Haan	20	Research
King	10	Accounting
Kochhar	10	Accounting

3 rows returned in 0.01 seconds Download

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

QUERY:

```
SELECT DISTINCT j.job_title, l.city FROM jobs j JOIN departments d ON j.min_salary <= d.department_budget AND j.max_salary >= d.department_budget JOIN locations l ON d.location_id = l.location_id WHERE d.department_id = 80;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

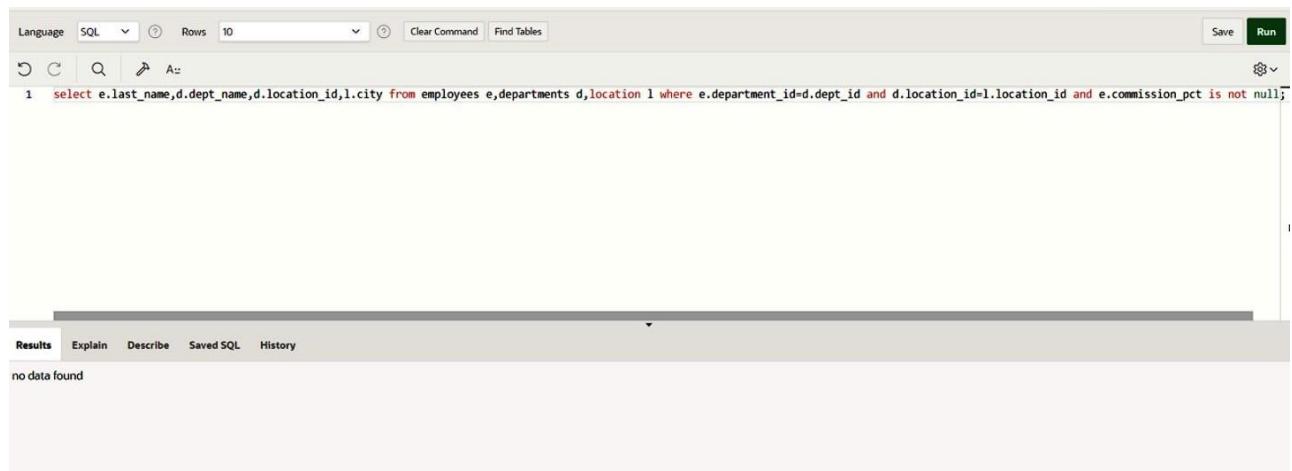
- Language:** SQL
- Rows:** 10
- Command:** `select distinct job_id,location_id from employees,departments where employees.department_id=departments.dept_id and employees.department_id=80;`
- Results:** The results table has two columns: **JOB_ID** and **LOCATION_ID**. It contains one row: ac_account with LOCATION_ID 4598.
- Timing:** 1 rows returned in 0.02 seconds

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

QUERY:

```
SELECT e.last_name, d.department_name, l.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;
```

OUTPUT



The screenshot shows a SQL query editor interface. The query entered is:

```
1 select e.last_name,d.dept_name,d.location_id,l.city from employees e,departments d,location l where e.department_id=d.dept_id and d.location_id=l.location_id and e.commission_pct is not null;
```

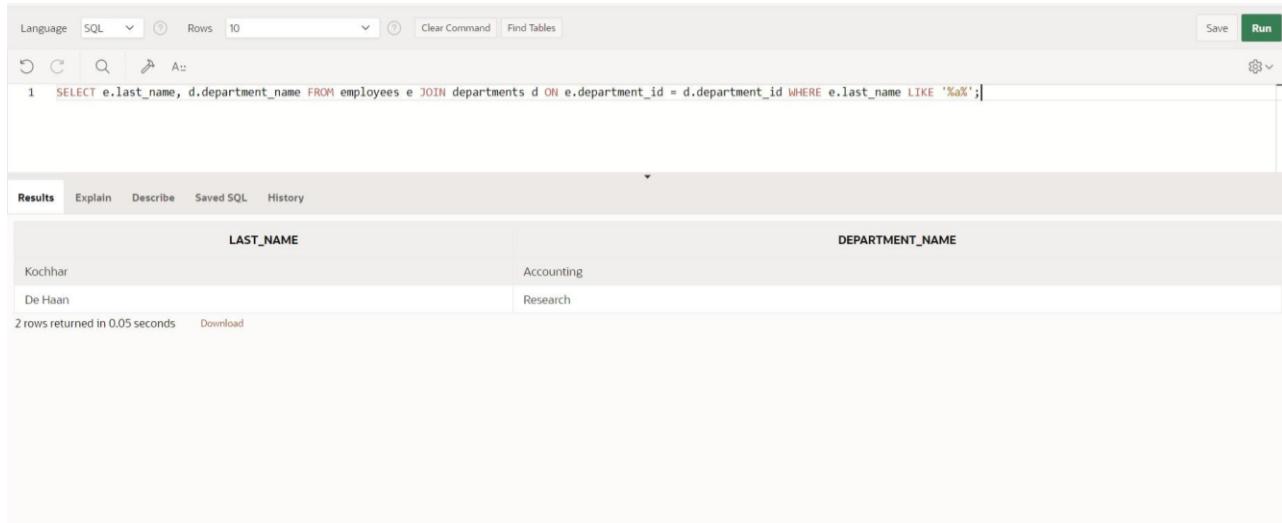
The results pane below shows the message "no data found".

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

QUERY:

```
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%';
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
1  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%';
```

The results table has two columns: LAST_NAME and DEPARTMENT_NAME. The data returned is:

LAST_NAME	DEPARTMENT_NAME
Kochhar	Accounting
De Haan	Research

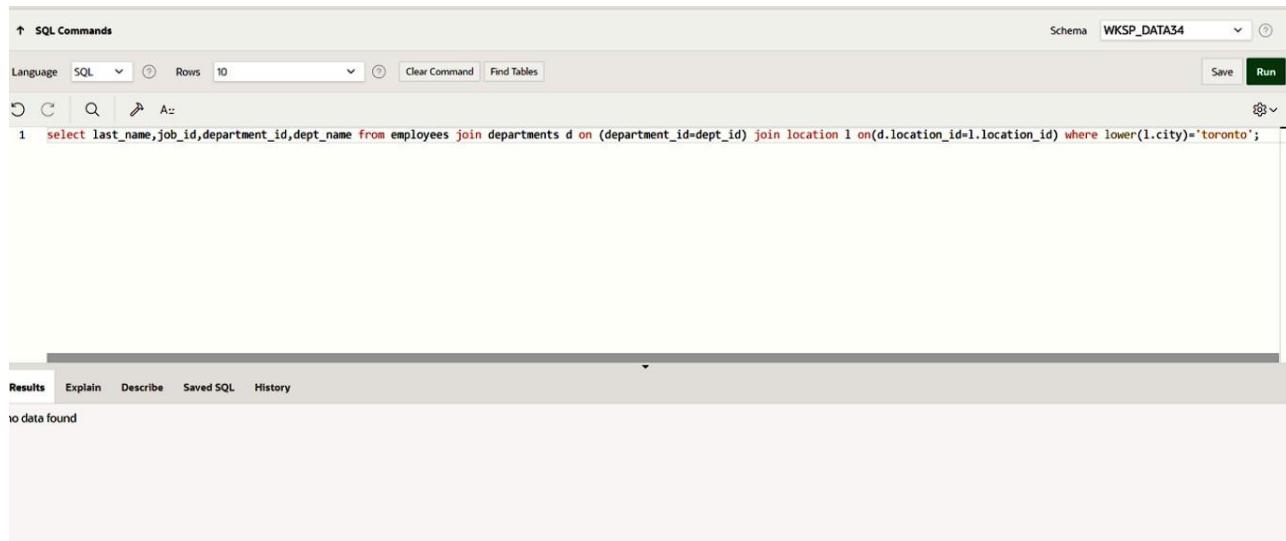
Below the table, it says "2 rows returned in 0.05 seconds".

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

QUERY:

```
SELECT e.last_name, j.job_title, e.department_id, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id JOIN jobs j ON e.job_id = j.job_id JOIN locations l ON d.location_id = l.location_id WHERE l.city = 'Toronto';
```

OUTPUT:



The screenshot shows a SQL command window interface. The top bar includes tabs for 'SQL Commands' (selected), 'Language' (set to 'SQL'), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Schema' (set to 'WKSP_DATA34'), and 'Run' (a green button). Below the toolbar, there are icons for refresh, undo, redo, search, and other utilities. The main area contains a single line of SQL code:

```
1 select last_name,job_id,department_id,dept_name from employees join departments d on (department_id=dept_id) join location l on(d.location_id=l.location_id) where lower(l.city)='toronto';
```

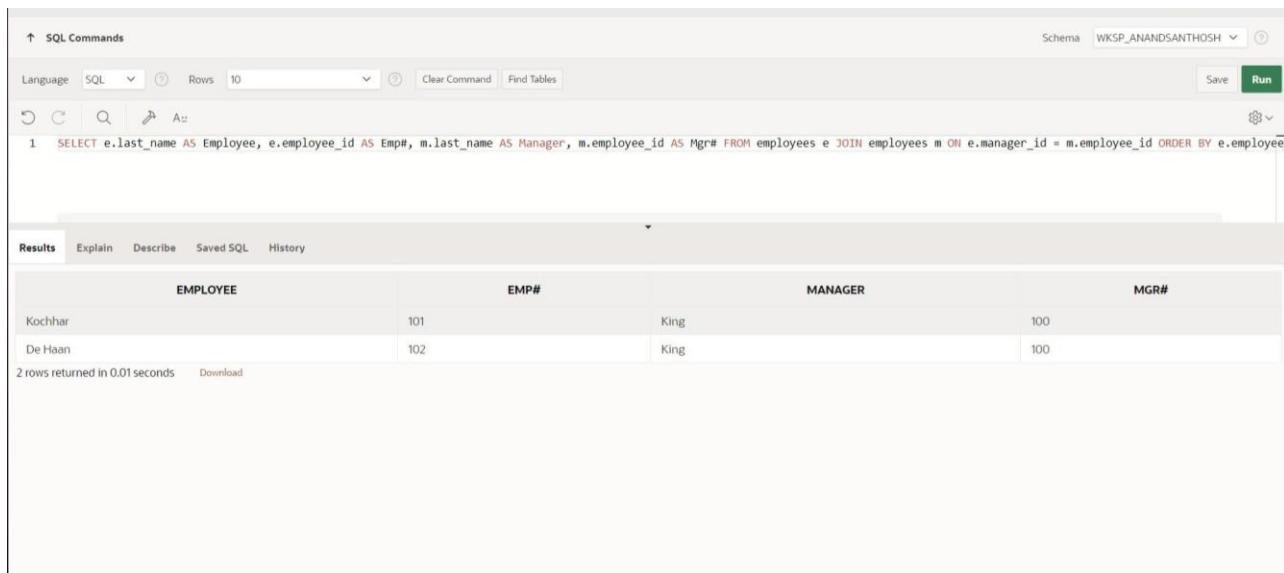
At the bottom, a navigation bar offers options: 'Results' (selected), 'Explain', 'Describe', 'Saved SQL', and 'History'. A message 'no data found' is displayed below the results area.

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

QUERY:

```
SELECT e.last_name AS Employee, e.employee_id AS Emp#, m.last_name AS Manager, m.employee_id AS Mgr# FROM employees e JOIN employees m ON e.manager_id = m.employee_id ORDER BY e.employee_id;
```

OUTPUT:



The screenshot shows a SQL command window with the following details:

- SQL Commands:** The query is displayed in the command input area.
- Results:** The results are presented in a tabular format with four columns: EMPLOYEE, EMP#, MANAGER, and MGR#.
- Data:** Two rows are returned, showing the following information:

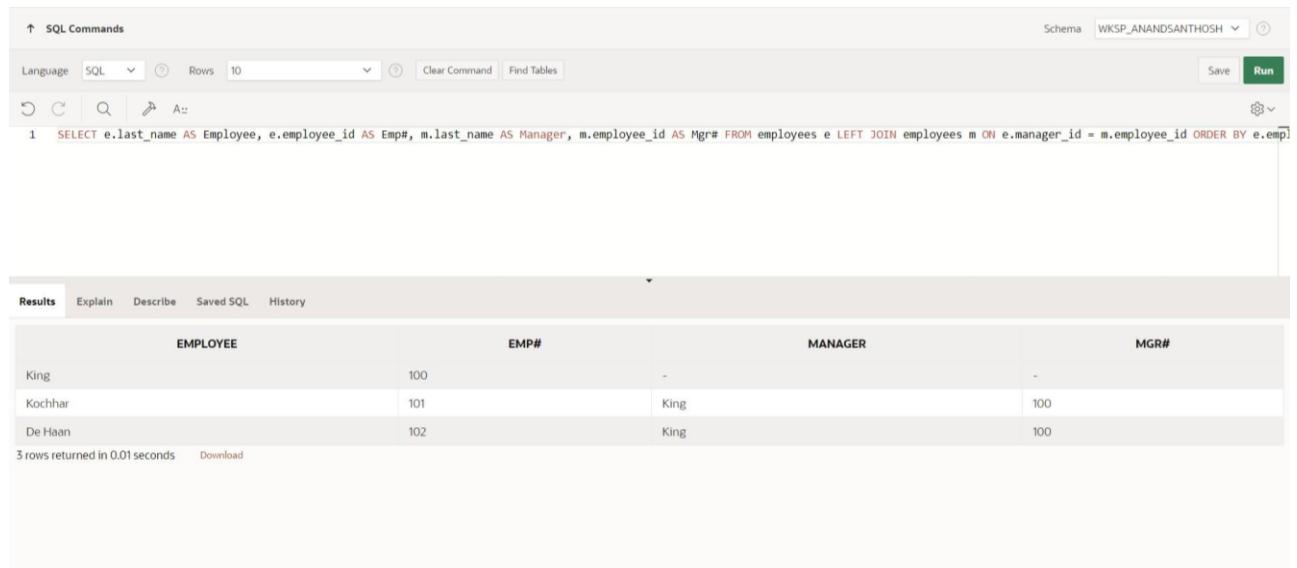
EMPLOYEE	EMP#	MANAGER	MGR#
Kochhar	101	King	100
De Haan	102	King	100
- Timing:** The query executed in 0.01 seconds.

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

QUERY:

```
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 ORDER BY e.employee_id;
```

OUTPUT:



The screenshot shows a SQL command window with the following details:

- Language: SQL
- Schema: WKSP_ANANDSANTHOSH
- Query: `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 ORDER BY e.employee_id;`
- Results tab selected.
- Table output:

EMPLOYEE	EMP#	MANAGER	MGR#
King	100	-	-
Kochhar	101	King	100
De Haan	102	King	100

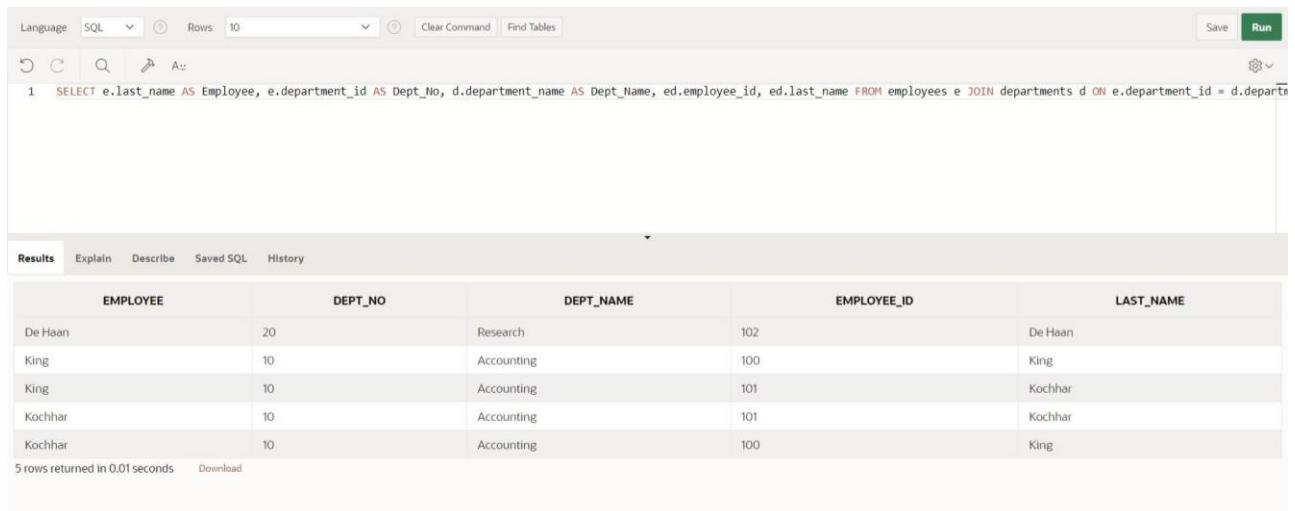
3 rows returned in 0.01 seconds Download

8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label

QUERY:

```
SELECT e.last_name AS Employee, e.department_id AS Dept_No, d.department_name AS Dept_Name, ed.employee_id, ed.last_name FROM employees e JOIN departments d ON e.department_id = d.department_id LEFT JOIN employees ed ON e.department_id = ed.department_id ORDER BY e.last_name;
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query is:

```
1 SELECT e.last_name AS Employee, e.department_id AS Dept_No, d.department_name AS Dept_Name, ed.employee_id, ed.last_name FROM employees e JOIN departments d ON e.department_id = d.department_id LEFT JOIN employees ed ON e.department_id = ed.department_id ORDER BY e.last_name;
```

The results table has the following data:

EMPLOYEE	DEPT_NO	DEPT_NAME	EMPLOYEE_ID	LAST_NAME
De Haan	20	Research	102	De Haan
King	10	Accounting	100	King
King	10	Accounting	101	Kochhar
Kochhar	10	Accounting	101	Kochhar
Kochhar	10	Accounting	100	King

5 rows returned in 0.01 seconds Download

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

QUERY:

```
DESCRIBE job_grades;
```

```
SELECT e.first_name || ' ' || e.last_name AS Name, j.job_title, d.department_name, e.salary, jg.grade_level FROM employees e JOIN jobs j ON e.job_id = j.job_id JOIN departments d ON e.department_id = d.department_id JOIN job_grades jg ON e.salary BETWEEN jg.lowest_sal AND jg.highest_sal;
```

OUTPUT:

The screenshot shows a SQL query results table. The columns are NAME, JOB_TITLE, DEPARTMENT_NAME, SALARY, and GRADE_LEVEL. The data includes Steven King (Administration President, Accounting, 24000, 6), Neena Kochhar (Administration Vice President, Accounting, 17000, 5), and Lex De Haan (Administration Vice President, Research, 17000, 5). A note at the bottom says "5 rows returned in 0.01 seconds".

NAME	JOB_TITLE	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
Steven King	Administration President	Accounting	24000	6
Neena Kochhar	Administration Vice President	Accounting	17000	5
Lex De Haan	Administration Vice President	Research	17000	5

The screenshot shows a SQL DESCRIBE command results table for the JOB_GRADES table. It lists three columns: GRADE_LEVEL, LOWEST_SAL, and HIGHEST_SAL. The GRADE_LEVEL column is of type NUMBER(4,0) and has a primary key constraint. The LOWEST_SAL and HIGHEST_SAL columns are of type NUMBER(8,2).

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
JOB_GRADES	GRADE_LEVEL	NUMBER	4	0	1	YES	NO	NO	
	LOWEST_SAL	NUMBER	8	2	0	NO	NO	NO	
	HIGHEST_SAL	NUMBER	8	2	0	NO	NO	NO	

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

QUERY:

```
SELECT e.first_name || ' ' || e.last_name AS Employee, e.hire_date FROM employees e WHERE e.hire_date > (SELECT e2.hire_date FROM employees e2 WHERE e2.last_name = 'Davies');
```

OUTPUT:



The screenshot shows a SQL query editor interface. The top bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and a 'Run' button. The main area contains the following SQL code:

```
1 SELECT e.last_name, e.hire_date
2 FROM employees e, employees davies
3 WHERE davies.last_name = 'Davies'
4 AND davies.hire_date < e.hire_date;
5
```

Below the code, the 'Results' tab is selected, showing the output table:

LAST_NAME	HIRE_DATE
Janu	03/05/2024

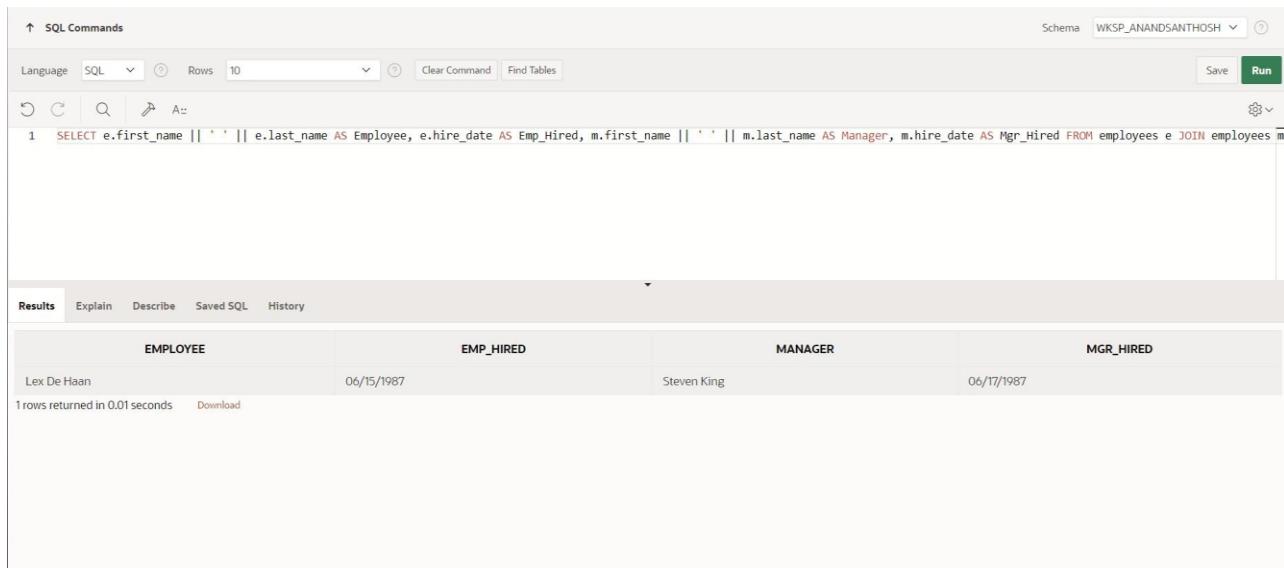
Below the table, it says '1 rows returned in 0.01 seconds' and has a 'Download' link.

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

QUERY:

```
SELECT e.first_name || ' ' || e.last_name AS Employee, e.hire_date AS Emp_Hired, m.first_name || '  
' || 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 ORDER BY e.hire_date;
```

OUTPUT:



The screenshot shows a SQL command window with the following details:

- Schema: WKSP_ANANDSANTHOSH
- Language: SQL
- Rows: 10
- Clear Command, Find Tables buttons
- Run button
- SQL Command input field containing the query: `1 SELECT e.first_name || ' ' || e.last_name AS Employee, e.hire_date AS Emp_Hired, m.first_name || '
' || 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 ORDER BY e.hire_date;`
- Results tab selected
- Table Headers: EMPLOYEE, EMP_HIRED, MANAGER, MGR_HIRED
- Table Data:

EMPLOYEE	EMP_HIRED	MANAGER	MGR_HIRED
Lex De Haan	06/15/1987	Steven King	06/17/1987
- Message: 1 rows returned in 0.01 seconds
- Download link

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

AGGREGATING DATA USING GROUP FUNCTIONS

EX.NO:8

DATE: 15 – 3 - 24

REG.NO: 220701239

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

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

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

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

The HR department needs the following reports:

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

QUERY:

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

OUTPUT:

The screenshot shows a MySQL command-line interface. The SQL tab contains the following query:

```
1 SELECT  
2     MAX(salary) AS "Maximum",  
3     MIN(salary) AS "Minimum",  
4     SUM(salary) AS "Sum",  
5     ROUND(AVG(salary)) AS "Average"  
6 FROM employees;
```

The Results tab displays the output:

	Maximum	Minimum	Sum	Average
Rows	90000	70000	407000	56100

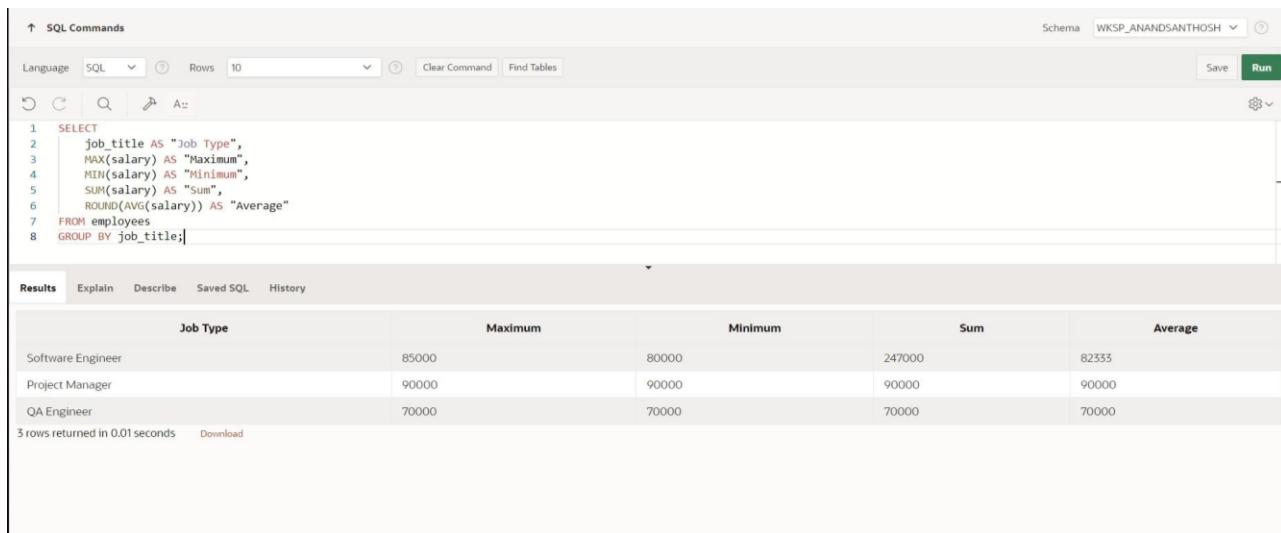
Time taken: 0.01 seconds. Download

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

QUERY:

```
SELECT job_title AS "Job Type", MAX(salary) AS "Maximum", MIN(salary) AS "Minimum",
SUM(salary) AS "Sum", ROUND(AVG(salary)) AS "Average" FROM employees GROUP BY
job_title;
```

OUTPUT:



The screenshot shows a SQL query editor interface with the following details:

- SQL Commands:** The query is pasted into the command input field.
- Language:** SQL
- Schema:** WKSP_ANANDSANHOSH
- Run:** The query has been run successfully.
- Results:** The results are displayed in a table format.
- Table Headers:** Job Type, Maximum, Minimum, Sum, Average.
- Table Data:**

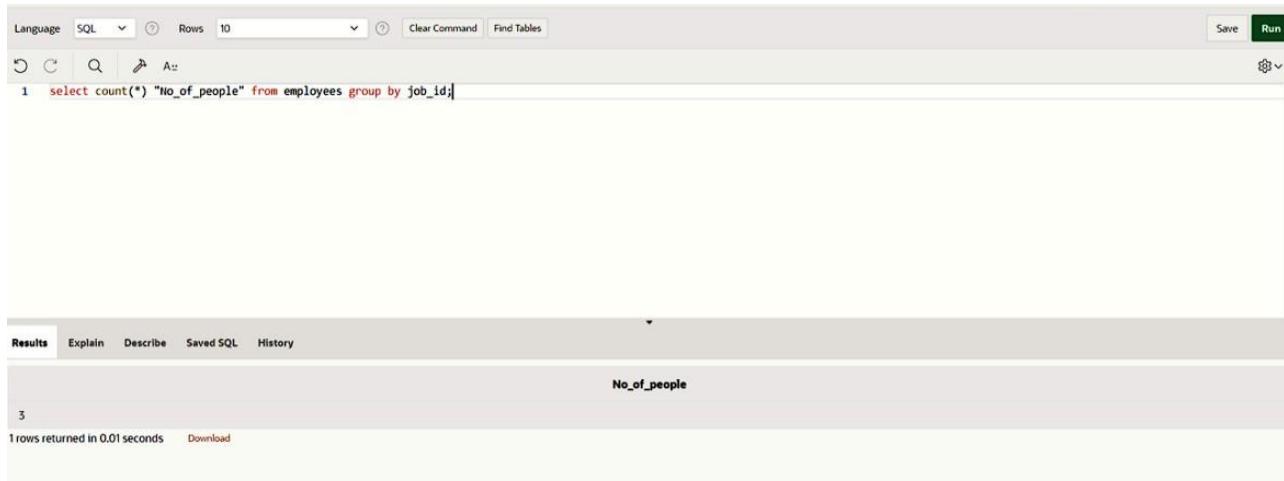
Job Type	Maximum	Minimum	Sum	Average
Software Engineer	85000	80000	247000	82333
Project Manager	90000	90000	90000	90000
QA Engineer	70000	70000	70000	70000
- Timing:** 3 rows returned in 0.01 seconds.

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

QUERY:

```
SELECT job_title, COUNT(*) AS "Number of Employees" FROM employees WHERE job_title = &job_title;
```

OUTPUT:



The screenshot shows a MySQL command-line interface window. The query entered is:

```
1 select count(*) "No_of_people" from employees group by job_id;
```

The results section shows a single row with the value '3' under the column 'No_of_people'. The status bar at the bottom indicates '1 rows returned in 0.01 seconds'.

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

QUERY:

```
SELECT COUNT(*) AS "Number of Managers" FROM employees WHERE manager_id IS NOT NULL;
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
select count(manager_id) "Number of Managers" from employees where manager_id is not null;
```

The results section displays the output:

Number of Managers
3

Below the table, it says "1 rows returned in 0.01 seconds".

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

QUERY:

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

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. The SQL command entered is:

```
1 SELECT
2 | MAX(salary) - MIN(salary) AS "DIFFERENCE"
3 FROM employees;
```

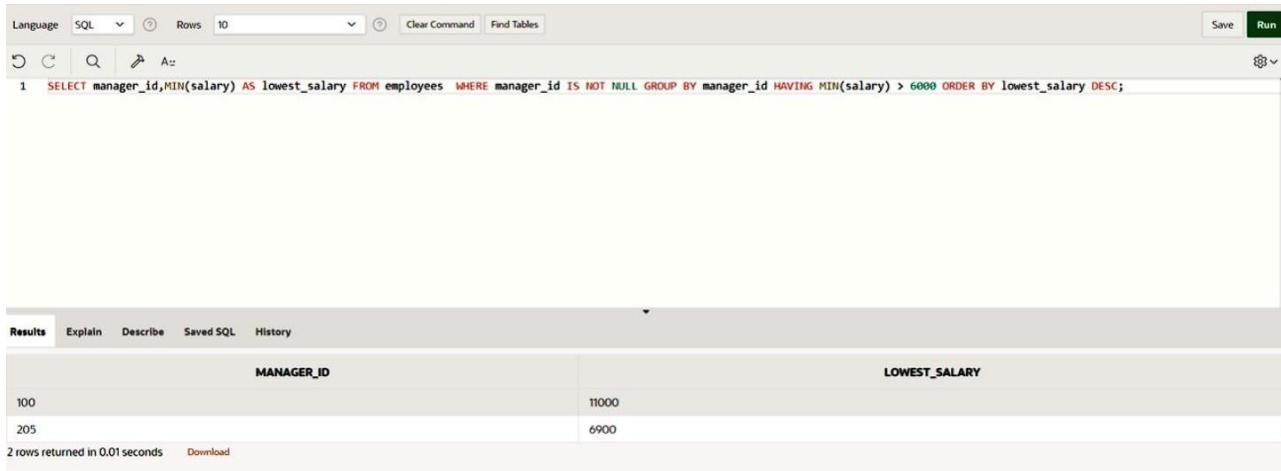
Below the command, the Results tab is selected. The output table has one row with the header "DIFFERENCE". The value in the row is 20000. Below the table, it says "1 rows returned in 0.00 seconds" and there is a Download button.

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

QUERY:

```
SELECT manager_id, MIN(salary) AS "Lowest Salary" FROM employees WHERE manager_id IS NOT NULL AND salary > 6000 GROUP BY manager_id ORDER BY MIN(salary) DESC;
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query is:

```
1 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;
```

The results table has two columns: MANAGER_ID and LOWEST_SALARY. The data is:

MANAGER_ID	LOWEST_SALARY
100	11000
205	6900

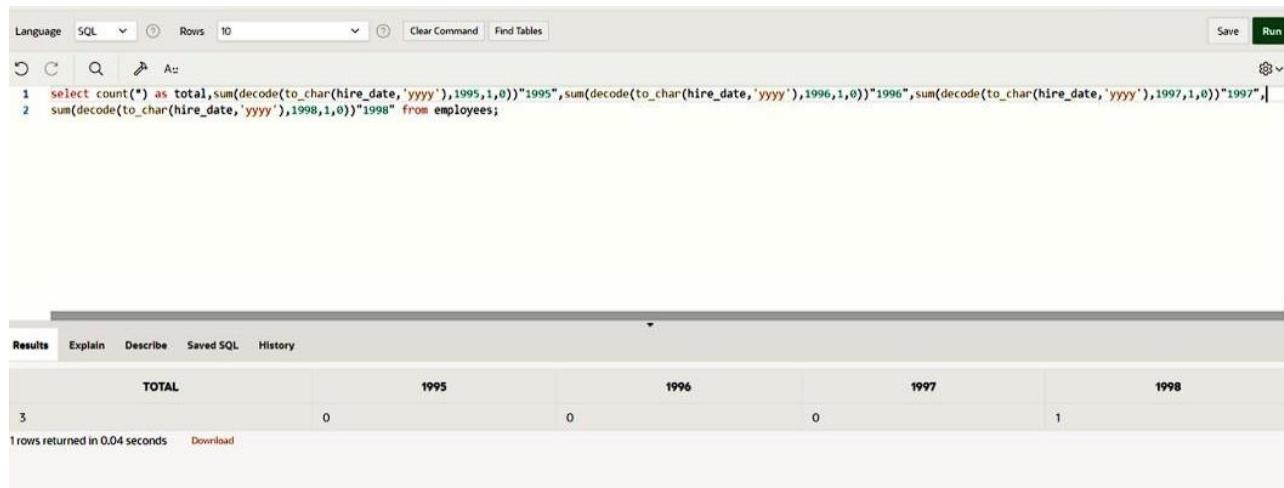
2 rows returned in 0.01 seconds [Download](#)

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

QUERY:

```
SELECT COUNT(*) AS "Total Employees",
SUM(CASE WHEN hire_date LIKE '1995%' THEN 1 ELSE 0 END) AS "Employees in 1995",
SUM(CASE WHEN hire_date LIKE '1996%' THEN 1 ELSE 0 END) AS "Employees in 1996",
SUM(CASE WHEN hire_date LIKE '1997%' THEN 1 ELSE 0 END) AS "Employees in 1997",
SUM(CASE WHEN hire_date LIKE '1998%' THEN 1 ELSE 0 END) AS "Employees in 1998"
FROM employees;
```

OUTPUT:



The screenshot shows a MySQL Workbench interface. The SQL tab contains the following query:

```
1 select count(*) as total,sum(decode(to_char(hire_date,'yyyy'),1995,1,0))"1995",sum(decode(to_char(hire_date,'yyyy'),1996,1,0))"1996",sum(decode(to_char(hire_date,'yyyy'),1997,1,0))"1997",
2 sum(decode(to_char(hire_date, 'yyyy'),1998,1,0))"1998" from employees;
```

The Results tab displays the output:

	TOTAL	1995	1996	1997	1998
3	0	0	0	0	1

1 rows returned in 0.04 seconds Download

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

QUERY:

```
SELECT d.department_name AS "Department", e.job_title AS "Job", e.salary AS "Salary",
SUM(e.salary) OVER (PARTITION BY e.job_title) AS "Total Salary"
FROM employees e JOIN departments d ON e.department_id = d.department_id
WHERE d.department_id IN (20, 50, 80, 90);
```

OUTPUT:

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

QUERY:

```
SELECT d.department_name AS "Location", d.location_id AS "Department", COUNT(*) AS
"Number of people", ROUND(AVG(e.salary), 2) AS "Salary" FROM employees e
JOIN departments d ON e.department_id = d.department_id GROUP BY d.department_name,
d.location ORDER BY "Salary" DESC;
```

OUTPUT:

The screenshot shows a SQL query editor interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Query Text:**

```
1 select d.dept_name as "Department name",l.location_id as "Location",count(e.department_id) as "Number of people",round(avg(e.salary),2) as "Salary"
2 from departments d,employees e,location l where d.dept_id=e.department_id group by d.dept_name,l.location_id,e.department_id;
```
- Results Tab:** Shows a table with the following data:

Department name	Location	Number of people	Salary
Public Relations	4598	1	6900
Public Relations	1231	1	6900
finance	4598	1	11000
finance	1231	1	11000

4 rows returned in 0.02 seconds Download

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

SUB QUERIES

EX.NO:9

DATE: 28 – 3 - 24

REG.NO: 220701239

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).

QUERY:

```
SELECT last_name, hire_date FROM employees  
WHERE department_id IN (SELECT department_id FROM employees WHERE last_name =  
:last_name) AND employee_id != (SELECT employee_id FROM employees WHERE last_name =  
:last_name);
```

OUTPUT:

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Bind Variables' dialog box with a single entry: 'LAST_NAME' with a value of 'Austin'. Below it, the main SQL editor window contains the following SQL code:

```
1 SELECT last_name, hire_date  
2 FROM employees  
3 WHERE department_id IN (SELECT department_id FROM employees WHERE last_name = :last_name)  
4 AND employee_id != (SELECT employee_id FROM employees WHERE last_name = :last_name);
```

At the bottom, the 'Results' tab displays the output of the query:

LAST_NAME	HIRE_DATE
Hunold	01/09/2006
Immler	05/21/2007
Ratabala	02/09/2006
Lorentz	02/07/2007

The results show four employees with the last name 'Austin' excluding themselves, with their hire dates.

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

QUERY:

```
SELECT employee_id, last_name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees) ORDER BY salary ASC;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (10), Clear Command, Find Tables, Save, and Run. Below the toolbar, the SQL code is displayed:

```
1 SELECT employee_id, last_name, salary
2 FROM employees
3 WHERE salary > (SELECT AVG(salary) FROM employees)
4 ORDER BY salary ASC;
```

Below the code, there are tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected, showing a table with three columns: EMPLOYEE_ID, LAST_NAME, and SALARY. The data returned is:

EMPLOYEE_ID	LAST_NAME	SALARY
115	Rajs	11000
109	Greenberg	12000
102	Kochhar	17000
103	De Haan	17000
101	King	24000

At the bottom left, it says "5 rows returned in 0.01 seconds".

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

QUERY:

```
SELECT employee_id, last_name
FROM employees
WHERE department_id IN (SELECT department_id FROM employees WHERE last_name LIKE
'%u%');
```

OUTPUT:

The screenshot shows a SQL query editor interface. The top bar includes 'Language' set to 'SQL', 'Rows' set to 10, and buttons for 'Save' and 'Run'. The main area displays the following SQL code:

```
1 SELECT employee_id, last_name
2 FROM employees
3 WHERE department_id IN (SELECT department_id FROM employees WHERE last_name LIKE
'%u%');
```

Below the code, there are tabs for 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is selected, showing a table with two columns: 'EMPLOYEE_ID' and 'LAST_NAME'. The data returned is:

EMPLOYEE_ID	LAST_NAME
104	Hunold
105	Ernst
106	Austin
107	Pataballa
108	Lorentz

At the bottom left, it says '5 rows returned in 0.01 seconds'. There is also a 'Download' link.

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

QUERY:

```
SELECT last_name, department_id, job_id FROM employees WHERE department_id IN  
(SELECT department_id FROM departments WHERE location_id = 1700);
```

OUTPUT:



The screenshot shows a SQL query editor interface. The top bar includes 'Language' set to 'SQL', 'Rows' set to '10', and buttons for 'Save' and 'Run'. The main area contains the SQL code:

```
1 select last_name,department_id,job_id from employees where department_id=(select dept_id from departments where location_id=1700);
```

The results tab is selected, displaying the following table:

LAST_NAME	DEPARTMENT_ID	JOB_ID
Janu	100	ac_account
Doe	100	ac_account

Below the table, it says '2 rows returned in 0.04 seconds' and has a 'Download' link.

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

QUERY:

```
SELECT last_name, salary  
FROM employees  
WHERE manager_id = (SELECT employee_id FROM employees WHERE last_name = 'King');
```

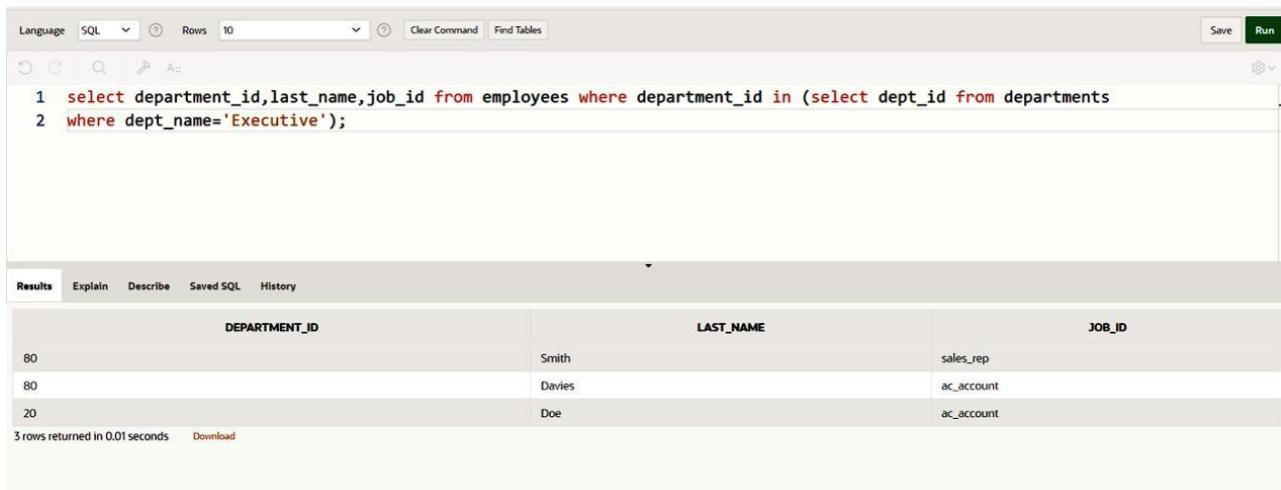
OUTPUT:

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

QUERY:

```
SELECT department_id, last_name, job_id  
FROM employees  
WHERE department_id IN (SELECT department_id FROM departments WHERE  
department_name = 'Executive');
```

OUTPUT:



The screenshot shows a SQL query editor interface. The top bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and 'Run' buttons. The main area contains the following SQL code:

```
1 select department_id, last_name, job_id from employees where department_id in (select dept_id from departments  
2 where dept_name='Executive');
```

The results section shows a table with three columns: DEPARTMENT_ID, LAST_NAME, and JOB_ID. The data is as follows:

DEPARTMENT_ID	LAST_NAME	JOB_ID
80	Smith	sales_rep
80	Davies	ac_account
20	Doe	ac_account

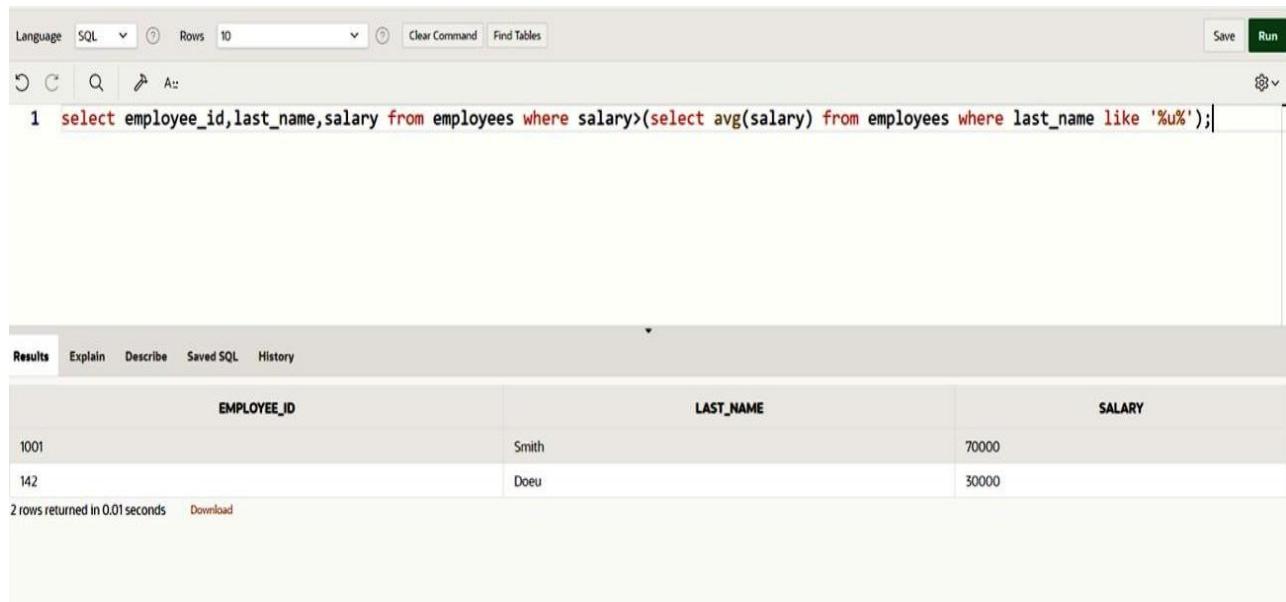
At the bottom left, it says '3 rows returned in 0.01 seconds'. There are also 'Download' and 'History' buttons.

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

QUERY:

```
SELECT employee_id, last_name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees) AND department_id IN (SELECT department_id FROM employees WHERE last_name LIKE "%u%");
```

OUTPUT:



The screenshot shows a SQL query editor interface. The top bar includes 'Language' set to 'SQL', 'Rows' set to '10', 'Clear Command', 'Find Tables', 'Save', and 'Run' buttons. Below the toolbar is a toolbar with icons for copy, paste, search, and refresh. The main area contains the SQL query:

```
1 select employee_id, last_name, salary from employees where salary > (select avg(salary) from employees where last_name like '%u%');
```

Below the query is a results table with three rows: 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The results table has columns 'EMPLOYEE_ID', 'LAST_NAME', and 'SALARY'. The data is as follows:

EMPLOYEE_ID	LAST_NAME	SALARY
1001	Smith	70000
142	Doeu	30000

At the bottom left, it says '2 rows returned in 0.01 seconds' and there is a 'Download' link.

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

USING THE SET OPERATORS

EX.NO:10

DATE: 04 – 4 - 24

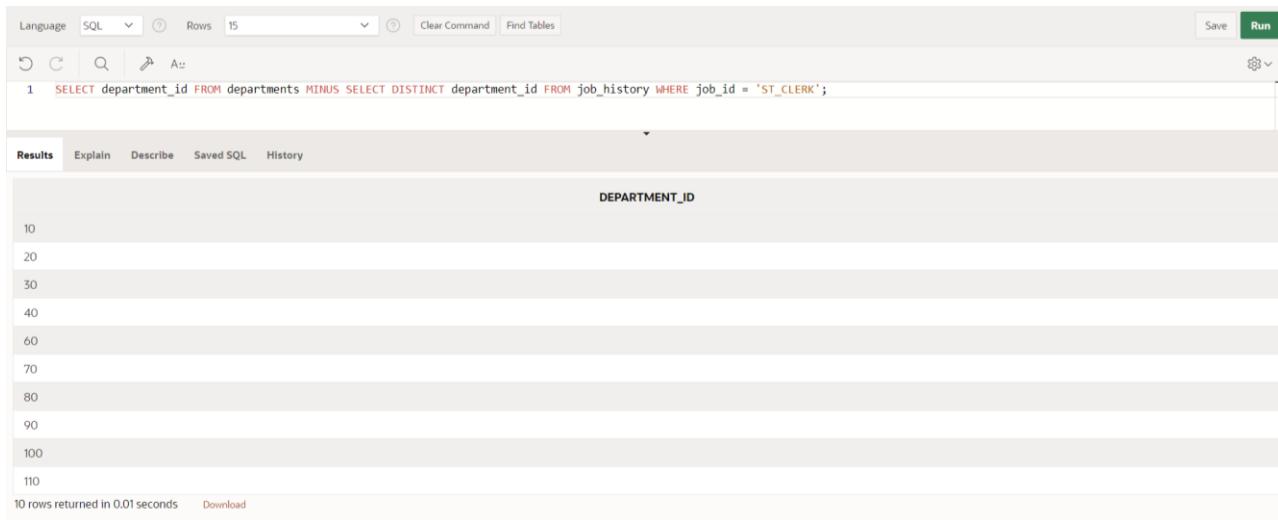
REG.NO: 220701239

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.

QUERY:

```
SELECT department_id FROM departments MINUS SELECT DISTINCT department_id FROM job_history WHERE job_id = 'ST_CLERK';
```

OUTPUT:



The screenshot shows a SQL query execution interface. The query entered is:

```
1  SELECT department_id FROM departments MINUS SELECT DISTINCT department_id FROM job_history WHERE job_id = 'ST_CLERK';
```

The results section displays the following data:

DEPARTMENT_ID
10
20
30
40
60
70
80
90
100
110

10 rows returned in 0.01 seconds Download

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

QUERY:

```
SELECT country_id, country_name FROM countries
MINUS
SELECT DISTINCT c.country_id, c.country_name FROM departments d
JOIN locations l ON d.location_id = l.location_id
JOIN countries c ON l.country_id = c.country_id;
```

OUTPUT:

The screenshot shows a SQL query editor interface. The query in the command window is:

```
1 SELECT country_id, country_name
2 FROM countries
3 MINUS
4 SELECT DISTINCT c.country_id, c.country_name
5 FROM departments d
6 JOIN locations l ON d.location_id = l.location_id
7 JOIN countries c ON l.country_id = c.country_id;
```

The results table has two columns: COUNTRY_ID and COUNTRY_NAME. The data is:

COUNTRY_ID	COUNTRY_NAME
CA	Canada
CN	China
DE	Germany
FR	France
IT	Italy
JP	Japan
MX	Mexico
UK	United Kingdom

8 rows returned in 0.02 seconds Download

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

department ID using set operators.

QUERY:

```
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;
```

OUTPUT:

The screenshot shows a SQL query editor interface. The top bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and 'Run' buttons. The main area displays the following SQL code:

```
1 select job_id,department_id from employees where department_id=10 union  
2 select job_id,department_id from employees where department_id=50 union  
3 select job_id,department_id from employees where department_id=20;
```

Below the code, the results tab is selected, showing a table with two columns: 'JOB_ID' and 'DEPARTMENT_ID'. The data returned is:

JOB_ID	DEPARTMENT_ID
ac_account	20
hr_rep	20

Text at the bottom of the results pane indicates "2 rows returned in 0.01 seconds" and a "Download" link.

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

their original job).

QUERY:

```
select job_id,employee_id from employees intersect select e.job_id,e.employee_id from employees  
e,job_history j where e.job_id=j.old_job_id;
```

OUTPUT:

The screenshot shows a SQL command window with the following details:

- SQL Commands:** The top bar includes "SQL Commands", "Schema: WKSP_DATA34", "Save", and "Run" buttons.
- Query:** The command entered is:

```
1 select job_id,employee_id from employees intersect select e.job_id,e.employee_id from employees e,job_history j where e.job_id=j.old_job_id;
```
- Results:** The results tab is selected, displaying the output of the query:

JOB_ID	EMPLOYEE_ID
ac_account	113
ac_account	142
sales_rep	1001

- Timing:** The message "3 rows returned in 0.03 seconds" is visible at the bottom left.

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

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

QUERY:

```
SELECT last_name, department_id FROM employees UNION SELECT department_name, department_id FROM departments;
```

OUTPUT:

SQL Commands	
Language SQL Rows 100 Clear Command Find Tables	
1 SELECT last_name, department_id FROM employees UNION SELECT department_name, department_id FROM departments;	
Results	Explain Describe Saved SQL History
LAST_NAME	DEPARTMENT_ID
Chen	100
Accounting	110
Administration	10
Austin	60
De Haan	90
Ernst	60
Executive	90
Faviet	100
Finance	100
Greenberg	100
Human Resources	40

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

USING THE SET OPERATORS

EX.NO:11

DATE: 05 – 4 - 24

REG.NO: 220701239

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.

QUERY:

```
CREATE VIEW employee_vu (employee_number, employee, department_number)
AS SELECT employee_id, first_name || ' ' || last_name AS employee, department_id AS
department_number
FROM employees;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (100), Clear Command, Find Tables, Save, and Run. Below the toolbar, there are icons for Undo, Redo, Search, and Paste. The main area contains the SQL code for creating a view:

```
1 CREATE VIEW employee_vu (employee_number, employee, department_number)
2 AS SELECT employee_id, first_name || ' ' || last_name AS employee, department_id AS
3 department_number
FROM employees;
```

At the bottom, there is a results pane with tabs for Results, Explain, Describe, Saved SQL, and History. The Results tab is selected, showing the message "View created." and a timestamp "0.04 seconds".

2. Display the contents of the EMPLOYEES_VU view.

QUERY:

```
SELECT * FROM employee_vu;
```

OUTPUT:

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 100
- Clear Command
- Find Tables
- Run button
- SQL command: `1 SELECT * FROM employee_vu;`
- Results tab selected
- Table structure:

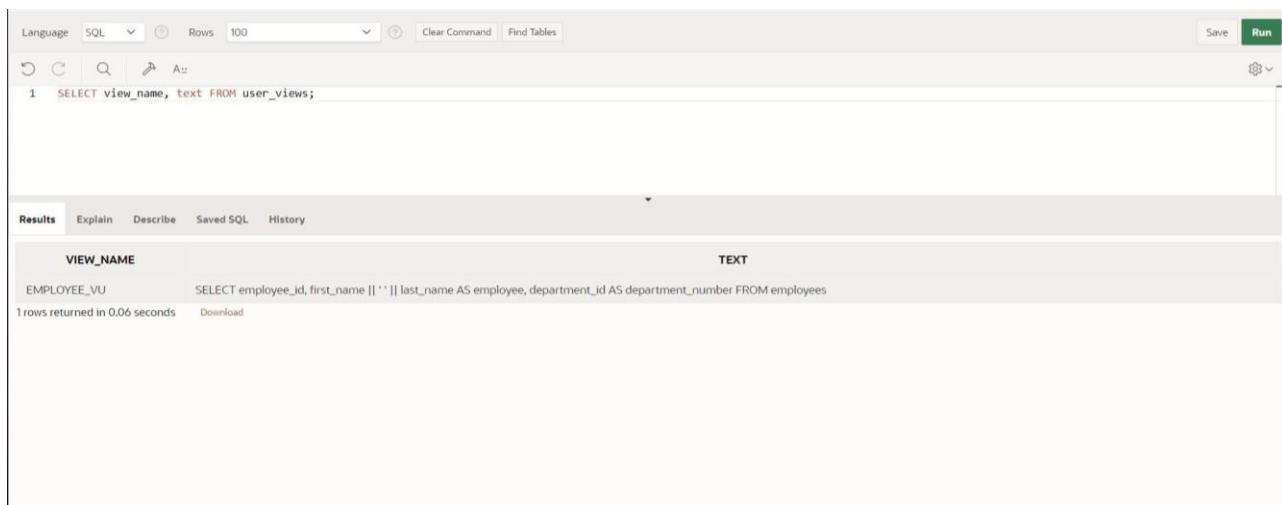
EMPLOYEE_NUMBER	EMPLOYEE	DEPARTMENT_NUMBER
102	Neena Kochhar	90
110	Daniel Faviet	100
107	Valli Pataballa	60
108	Diana Lorentz	60
111	John Chen	100
106	David Austin	60
103	Lex De Haan	90
105	Bruce Ernst	60
109	Nancy Greenberg	100
101	Steven King	90
104	Alexander Hunold	60
112	Ismael Scott	100

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

QUERY:

```
SELECT view_name, text FROM user_views;
```

OUTPUT:



The screenshot shows a SQL query interface with the following details:

- Query Bar:** Shows the SQL command: `SELECT view_name, text FROM user_views;`
- Results Tab:** Active tab, showing the output of the query.
- Output Headers:** `VIEW_NAME` and `TEXT`.
- Output Data:**

VIEW_NAME	TEXT
EMPLOYEE_VU	SELECT employee_id, first_name ' ' last_name AS employee, department_id AS department_number FROM employees
- Timing:** 1 rows returned in 0.06 seconds.
- Download:** Option to download the results.

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

QUERY:

```
SELECT employee, department_number FROM employee_vu;
```

OUTPUT:

The screenshot shows a SQL query interface with the following details:

- Language: SQL
- Rows: 100
- Clear Command | Find Tables
- Run button
- Results tab selected
- SQL command: `1 SELECT employee, department_number FROM employee_vu;`
- Table output:

EMPLOYEE	DEPARTMENT_NUMBER
Neena Kochhar	90
Daniel Faviet	100
Valli Pataballa	60
Diana Lorentz	60
John Chen	100
David Austin	60
Lex De Haan	90
Bruce Ernst	60
Nancy Greenberg	100
Steven King	90
Alexander Hunold	60
Ismael Scott	100

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

QUERY:

```
CREATE VIEW dept50 (empno, employee, deptno)
AS SELECT employee_id, last_name, department_id
FROM employees
WHERE department_id = 50
WITH CHECK OPTION;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are buttons for Language (SQL), Rows (100), Clear Command, Find Tables, Save, and Run. Below the toolbar, there are icons for refresh, undo, redo, search, and copy/paste. The main area displays the SQL code for creating the view:

```
1 CREATE VIEW dept50 (empno, employee, deptno)
2 AS SELECT employee_id, last_name, department_id
3 FROM employees
4 WHERE department_id = 50
5 WITH CHECK OPTION;
```

Below the code, there is a results tab labeled "Results". The output message "View created." is displayed, along with a timestamp "0.03 seconds".

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

QUERY:

```
DESCRIBE dept50;
```

OUTPUT:

The screenshot shows a database interface with a command line at the top and a results table below. The command entered is 'DESCRIBE dept50;'. The results table has tabs for 'Results', 'Explain', 'Describe' (which is selected), 'Saved SQL', and 'History'. The 'Describe' tab shows the structure of the VIEW 'DEPT50'. The table has columns: Table, Column, Data Type, Length, Precision, Scale, Primary Key, Nullable, Default, and Comment. The data for 'DEPT50' is as follows:

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
DEPT50	EMPNO	NUMBER	-	6	0	-	✓	-	-
	EMPLOYEE	VARCHAR2	25	-	-	-	-	-	-
	DEPTNO	NUMBER	-	4	0	-	✓	-	-

7. Attempt to reassign Matos to department 80.

QUERY:

```
UPDATE dept50 SET deptno = 80 WHERE empno = 201;
```

OUTPUT:

The screenshot shows a SQL query interface with the following details:

- Language:** SQL
- Rows:** 100
- Command:** UPDATE dept50 SET deptno = 80 WHERE empno = 201;
- Results:** 0 row(s) updated.
- Time:** 0.05 seconds

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

QUERY:

```
create or replace view salary_vu as select e.last_name "Employee",d.dept_name
Department, e.salary "Salary",j.grade_level "Grades" from employees e,departments
d,job_grade j where e.department_id=d.dept_id and e.salary between j.lowest_sal and
j.highest_sal;
```

OUTPUT:

The screenshot shows a SQL query being run in a database interface. The query is:

```
1 create or replace view salary_vu as
2 select e.last_name "Employee",d.dept_name "Department",e.salary "Salary",j.grade_level "Grades"
3 from employees e,departments d,job_grade j
4 where e.department_id=d.dept_id and e.salary between j.lowest_sal and j.highest_sal;
```

The results pane shows the message "View created." and a execution time of "0.06 seconds".

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

INTRO TO CONSTRAINTS: NOT NULL AND UNIQUE CONSTRAINTS

EX.NO:12

DATE: 12 – 4 - 24

REG.NO: 220701239

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store's locations. The owners want to make sure that all entries have an identification number, date opened, address, and city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global_locations table. Use the table for your answers.

Global Fast Foods global_locations Table						
NAME	TYPE	LENGTH	PRECISION	SCALE	NULLABLE	DEFAULT
Id						
name						
date_opened						
address						
city						
zip/postal code						
phone						
email						
manager_id						
Emergency contact						

1. What is a “constraint” as it relates to data integrity?

Database can be as reliable as the data in it, and database rules are implemented as Constraint to maintain data integrity.

2. What are the limitations of constraints that may be applied at the column level and at the table level?

- Constraints referring to more than one column are defined at Table Level
- NOT NULL constraint must be defined at column level as per ANSI/ISO SQL standard.

3. Why is it important to give meaningful names to constraints?

- If a constraint is violated in a SQL statement execution, it is easy to identify the cause with user-named constraints.
- It is easy to alter names/drop constraint.

4. Based on the information provided by the owners, choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.

Global Fast Foods global_locations Table						
NAME	TYPE	DataType	LENGTH	PRECISION	SCALE	NULLABLE
id	pk	NUMBER	6	0		No
name		VARCHAR2	50			
date_opened		DATE				No
address		VARCHAR2	50			No
city		VARCHAR2	30			No
zip_postal_code		VARCHAR2	12			
phone		VARCHAR2	20			
email	uk	VARCHAR2	75			
manager_id		NUMBER	6	0		
emergency_contact		VARCHAR2	20			

5. Use "(nullable)" to indicate those columns that can have null values.

Global Fast Foods global_locations Table						
NAME	TYPE	DataType	LENGTH	PRECISION	SCALE	NULLABLE
id	pk	NUMBER	6	0		No
name		VARCHAR2	50			Yes
date_opened		DATE				No
address		VARCHAR2	50			No
city		VARCHAR2	30			No
zip_postal_code		VARCHAR2	12			Yes
phone		VARCHAR2	20			Yes
email	uk	VARCHAR2	75			Yes
manager_id		NUMBER	6	0		Yes
emergency_contact		VARCHAR2	20			Yes

6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

```

CREATE TABLE f_global_locations
( id NUMBER(6,0) CONSTRAINT f_gln_id_pk PRIMARY KEY ,
  name VARCHAR2(50),
  date_opened DATE CONSTRAINT f_gln_dt_opened_nn NOT NULL ENABLE,
  address VARCHAR2(50) CONSTRAINT f_gln_add_nn NOT NULL ENABLE,
  city VARCHAR2(30) CONSTRAINT f_gln_city_nn NOT NULL ENABLE,
  zip_postal_code VARCHAR2(12),
  phone VARCHAR2(20),
  email VARCHAR2(75) CONSTRAINT f_gln_email_uk UNIQUE,
  manager_id NUMBER(6,0),
  emergency_contact VARCHAR2(20)
);

```

7. Execute the CREATE TABLE statement in Oracle Application Express.

Table Created.

8. Execute a DESCRIBE command to view the Table Summary information.

```
DESCRIBE f_global_locations;
```

9. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement.

NAME	TYPE	LENGTH	PRECISION	SCALE	NULLABLE	DEFAULT
id	number	4				
loc_name	varchar2	20			X	
	date					
address	varchar2	30				
city	varchar2	20				
zip_postal	varchar2	20			X	
phone	varchar2	15			X	
email	varchar2	80			X	
manager_id	number	4			X	
contact	varchar2	40			X	

```
CREATE TABLE f_global_locations
( id NUMBER(6,0) CONSTRAINT f_gln_id_pk PRIMARY KEY ,
name VARCHAR2(50),
date_opened DATE CONSTRAINT f_gln_dt_opened_nn NOT NULL ENABLE,
address VARCHAR2(50) CONSTRAINT f_gln_add_nn NOT NULL ENABLE,
city VARCHAR2(30) CONSTRAINT f_gln_city_nn NOT NULL ENABLE,
zip_postal_code VARCHAR2(12),
phone VARCHAR2(20),
email VARCHAR2(75),
manager_id NUMBER(6,0),
emergency_contact VARCHAR2(20),
CONSTRAINT f_gln_email_uk UNIQUE(email)
);
```

PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

1. What is the purpose of a
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK CONSTRAINT
- a. PRIMARY KEY Uniquely identify each row in table.
- b. FOREIGN KEY Referential integrity constraint links back parent table's primary/unique key to child table's column.
- c. CHECK CONSTRAINT Explicitly define condition to be met by each row's fields. This condition must be returned as true or unknown.
2. Using the column information for the animals table below, name constraints where applicable at the table level, otherwise name them at the column level. Define the primary key (animal_id). The license_tag_number must be unique. The admit_date and vaccination_date columns cannot contain null values.

animal_id NUMBER(6)
name VARCHAR2(25)
license_tag_number NUMBER(10)
admit_date DATE
adoption_id NUMBER(5),
vaccination_date DATE

animal_id NUMBER(6) - PRIMARY KEY

name VARCHAR2(25) license_tag_number NUMBER(10) - UNIQUE

admit_date DATE -NOT NULL

adoption_id NUMBER(5), vaccination_date DATE -NOT NULL

3. Create the animals table. Write the syntax you will use to create the table.

```
CREATE TABLE animals ( animal_id NUMBER(6,0) CONSTRAINT anl_anl_id_pk PRIMARY KEY , name  
VARCHAR2(25), license_tag_number NUMBER(10,0) CONSTRAINT anl_l_tag_num_uk UNIQUE,  
admit_date DATE CONSTRAINT anl_adt_dat_nn NOT NULL ENABLE, adoption_id NUMBER(5,0),  
vaccination_date DATE CONSTRAINT anl_vcc_dat_nn NOT NULL ENABLE );
```

4. Enter one row into the table. Execute a SELECT * statement to verify your input. Refer to the graphic below for input.

ANIMAL_ID	NAME	LICENSE_TAG_NUMBER	ADMIT_DATE	ADOPTION_ID	VACCINATION_DATE
101	Spot	35540	10-Oct-2004	205	12-Oct-2004

```
INSERT INTO animals (animal_id, name, license_tag_number, admit_date, adoption_id, vaccination_date) VALUES( 101, 'Spot', 35540, TO_DATE('10-Oct-2004', 'DD-Mon-YYYY'), 205, TO_DATE('12-Oct-2004', 'DD-Mon- YYYY'));
```

5. Write the syntax to create a foreign key (adoption_id) in the animals table that has a corresponding primary-key reference in the adoptions table. Show both the column-level and table-level syntax. Note that because you have not actually created an adoptions table, no adoption_id primary key exists, so the foreign key cannot be added to the animals table.

COLUMN LEVEL STATEMENT: ALTER TABLE animals MODIFY (adoption_id NUMBER(5,0) CONSTRAINT anl_adopt_id_fk REFERENCES adoptions(id) ENABLE);

TABLE LEVEL STATEMENT: ALTER TABLE animals ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id) REFERENCES adoptions(id) ENABLE;

6. What is the effect of setting the foreign key in the ANIMAL table as:

- a. ON DELETE CASCADE
- b. ON DELETE SET NULL

a. ON DELETE CASCADE ALTER TABLE animals ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id) REFERENCES adoptions(id) ON DELETE CASCADE ENABLE ;

b. ON DELETE SET NULL ALTER TABLE animals ADD CONSTRAINT anl_adopt_id_fk FOREIGN KEY (adoption_id) REFERENCES adoptions(id) ON DELETE SET NULL ENABLE ;

7. What are the restrictions on defining a CHECK constraint?

- I cannot specify check constraint for a view however in this case I could use WITH CHECK OPTION clause
- I am restricted to columns from self table and fields in self row.
- I cannot use subqueries and scalar subquery expressions.

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

CREATING VIEWS

EX.NO:13

DATE: 13 – 4 - 24

REG.NO: 220701239

1. What are three uses for a view from a DBA's perspective?

- Restrict access and display selective columns
- Reduce complexity of queries from other internal systems. So, providing a way to view same data in a different manner.
- Let the app code rely on views and allow the internal implementation of tables to be modified later

2. Create a simple view called view_d_songs that contains the ID, title and artist from the DJs on Demand table for each "New Age" type code. In the subquery, use the alias "Song Title" for the title column.

```
CREATE VIEW view_d_songs AS SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code where d_types.description = 'New Age';
```

3. SELECT * FROM view_d_songs. What was returned?

Results	Explain	Describe	Saved SQL	History
ID	Song Title		ARTIST	
47	Hurrah for Today		The Jubilant Trio	
49	Lets Celebrate		The Celebrants	

2 rows returned in 0.00 seconds [Download](#)

4. REPLACE view_d_songs. Add type_code to the column list. Use aliases for all columns. Or use alias after the CREATE statement as shown.

```
CREATE OR REPLACE VIEW view_d_songs AS SELECT d_songs.id, d_songs.title "Song Title", d_songs.artist, d_songs.type_code from d_songs INNER JOIN d_types ON d_songs.type_code = d_types.code where d_types.description = 'New Age';
```

5. Jason Tsang, the disk jockey for DJs on Demand, needs a list of the past events and those planned for the coming months so he can make arrangements for each event's equipment setup. As the company manager, you do not want him to have access to the price that clients paid for their events. Create a view for Jason to use that displays the name of the event, the event date, and the theme description. Use aliases for each column name.

```
CREATE OR REPLACE VIEW view_d_events_pkgs AS SELECT evt.name "Name of Event", TO_CHAR(evt.event_date, 'dd-Month-yyyy') "Event date", thm.description "Theme description" FROM d_events evt INNER JOIN d_themes thm ON evt.theme_code = thm.code WHERE evt.event_date <= ADD_MONTHS(SYSDATE,1);
```

6. It is company policy that only upper-level management be allowed access to individual employee salaries. The department managers, however, need to know the minimum, maximum, and average salaries, grouped by department. Use the Oracle database to prepare a view that displays the needed information for department managers.

```
CREATE OR REPLACE VIEW view_min_max_avg_dpt_salary ("Department Id", "Department Name", "Max Salary", "Min Salary", "Average Salary") AS SELECT dpt.department_id, dpt.department_name, MAX(NVL(emp.salary,0)), MIN(NVL(emp.salary,0)), ROUND(AVG(NVL(emp.salary,0)),2) FROM departments dpt LEFT OUTER JOIN employees emp ON dpt.department_id = emp.department_id GROUP BY (dpt.department_id, dpt.department_name);
```

DML OPERATIONS AND VIEWS

Use the DESCRIBE statement to verify that you have tables named copy_d_songs, copy_d_events, copy_d_cds, and copy_d_clients in your schema. If you don't, write a query to create a copy of each.

1. Query the data dictionary USER_UPDATABLE_COLUMNS to make sure the columns in the base tables will allow UPDATE, INSERT, or DELETE. All table names in the data dictionary are stored in uppercase. Use the same syntax but change table_name of the other tables.

```
SELECT owner, table_name, column_name, updatable,insertable, deletable FROM  
user_updatable_columns WHERE LOWER(table_name) = 'copy_d_songs';
```

```
SELECT owner, table_name, column_name, updatable,insertable, deletable FROM  
user_updatable_columns WHERE LOWER(table_name) = 'copy_d_events';
```

```
SELECT owner, table_name, column_name, updatable,insertable, deletable FROM  
user_updatable_columns WHERE LOWER(table_name) = 'copy_d_cds';
```

2. Use the CREATE or REPLACE option to create a view of *all* the columns in the copy_d_songs table called view_copy_d_songs.

```
CREATE OR REPLACE VIEW view_copy_d_songs AS SELECT * FROM  
copy_d_songs; SELECT * FROM view_copy_d_songs;
```

3. Use view_copy_d_songs to INSERT the following data into the underlying copy_d_songs table. Execute a SELECT * from copy_d_songs to verify your DML command. See the graphic.

ID	TITLE	DURATION	ARTIST	TYPE_CODE
88	Mello Jello	2	The What	4

```
INSERT INTO view_copy_d_songs(id,title,duration,artist,type_code) VALUES(88,'Mello Jello','2 min','The  
What',4);
```

4. Create a view based on the DJs on Demand COPY_D_CDS table. Name the view read_copy_d_cds. Select all columns to be included in the view. Add a WHERE clause to restrict the year to 2000. Add the WITH READ ONLY option.

```
CREATE OR REPLACE VIEW read_copy_d_cds AS SELECT * FROM copy_d_cds WHERE year = '2000' WITH  
READ ONLY ; SELECT * FROM read_copy_d_cds;
```

5. Using the read_copy_d_cds view, execute a DELETE FROM read_copy_d_cds WHERE cd_number = 90;

ORA-42399: cannot perform a DML operation on a read-only view

6. Use REPLACE to modify read_copy_d_cds. Replace the READ ONLY option with WITH CHECK OPTION CONSTRAINT ck_read_copy_d_cds. Execute a SELECT * statement to verify that the view exists.

```
CREATE OR REPLACE VIEW read_copy_d_cds AS SELECT * FROM copy_d_cds WHERE year = '2000' WITH  
CHECK OPTION CONSTRAINT ck_read_copy_d_cds;
```

7. Use the read_copy_d_cds view to delete any CD of year 2000 from the underlying copy_d_cds.

```
DELETE FROM read_copy_d_cds WHERE year = '2000';
```

8. Use the read_copy_d_cds view to delete cd_number 90 from the underlying copy_d_cds table.

```
DELETE FROM read_copy_d_cds WHERE cd_number = 90;
```

9. Use the read_copy_d_cds view to delete year 2001 records.

```
DELETE FROM read_copy_d_cds WHERE year = '2001';
```

10. Execute a SELECT * statement for the base table copy_d_cds. What rows were deleted?

Only the one in problem 7 above, not the one in 8 and 9

11. What are the restrictions on modifying data through a view?

DELETE,INSERT,MODIFY restricted if it contains:

Group functions

GROUP BY

CLAUSE DISTINCT

pseudocolumn ROWNUM Keyword

12. What is Moore's Law? Do you consider that it will continue to apply indefinitely? Support your opinion with research from the internet.

It roughly predicted that computing power nearly doubles every year. But Moore also said in 2005 that as per nature of exponential functions, this trend may not continue forever.

13. What is the "singularity" in terms of computing?

Singularity is the hypothesis that the invention of artificial superintelligence will abruptly trigger runaway technological growth, resulting in unfathomable changes to human civilization

1. Create a view from the copy_d_songs table called view_copy_d_songs that includes only the title and artist. Execute a SELECT * statement to verify that the view exists.

```
CREATE OR REPLACE VIEW view_copy_d_songs ASSELECT title, artistFROM copy_d_songs;SELECT * FROM view_copy_d_songs;
```

2. Issue a DROP view_copy_d_songs. Execute a SELECT * statement to verify that the view has been deleted.

```
DROP VIEW view_copy_d_songs; SELECT * FROM view_copy_d_songs;
```

ORA-00942: table or view does not exist

3. Create a query that selects the last name and salary from the Oracle database. Rank the salaries from highest to lowest for the top three employees.

```
SELECT * FROM(SELECT last_name, salary FROM employees ORDER BY salary DESC)WHERE ROWNUM <= 3;
```

4. Construct an inline view from the Oracle database that lists the last name, salary, department ID, and maximum salary for each department. Hint: One query will need to calculate maximum salary by department ID.

```
SELECT empm.last_name, empm.salary, dptmx.department_idFROM(SELECT dpt.department_id, MAX(NVL(emp.salary,0)) max_dpt_salFROM departments dpt LEFT OUTER JOIN employees emp ON dpt.department_id = emp.department_idGROUP BY dpt.department_id) dptmx LEFT OUTER JOIN employees empm ON dptmx.department_id = empm.department_idWHERE NVL(empm.salary,0) = dptmx.max_dpt_sal;
```

5. Create a query that will return the staff members of Global Fast Foods ranked by salary from lowest to highest.

```
SELECT ROWNUM,last_name,salaryFROM(SELECT * FROM f_staffs ORDER BY SALARY);
```

Indexes and Synonyms

1. What is an index and what is it used for?

Definition: These are schema objects which make retrieval of rows from table faster.

Purpose: An index provides direct and fast access to row in table. They provide indexed path to locate data quickly, so hereby reduce necessity of heavy disk input/output operations.

2. What is a ROWID, and how is it used?

Indexes use ROWID's (base 64 string representation of the row address containing block identifier, row location in the block and the database file identifier) which is the fastest way to access any particular row.

3. When will an index be created automatically?

Primary key/unique key use already existing unique index but if index is not present already, it is created while applying unique/primary key constraint.

4. Create a nonunique index (foreign key) for the DJs on Demand column (cd_number) in the D_TRACK_LISTINGS table. Use the Oracle Application Express SQL Workshop Data Browser to confirm that the index was created.

```
CREATE INDEX d_tlg_cd_number_fk_i ON d_track_listings (cd_number);
```

5. Use the join statement to display the indexes and uniqueness that exist in the data dictionary for the DJs on Demand D_SONGS table.

```
SELECT ucm.index_name, ucm.column_name, ucm.column_position, uix.uniqueness FROM user_indexes
  uix INNER JOIN user_ind_columns ucm ON uix.index_name = ucm.index_name WHERE ucm.table_name
= 'D_SONGS';
```

6. Use a SELECT statement to display the index_name, table_name, and uniqueness from the data dictionary USER_INDEXES for the DJs on Demand D_EVENTS table.

```
SELECT index_name, table_name, uniqueness FROM user_indexes WHERE table_name = 'D_EVENTS';
```

7. Write a query to create a synonym called dj_tracks for the DJs on Demand d_track_listings table.

```
CREATE SYNONYM dj_tracks FOR d_track_listings;
```

8. Create a function-based index for the last_name column in DJs on Demand D_PARTNERS table

that makes it possible not to have to capitalize the table name for searches. Write a SELECT statement that would use this index.

```
CREATE INDEX d_ptr_last_name_idx ON d_partners(LOWER(last_name));
```

9. Create a synonym for the D_TRACK_LISTINGS table. Confirm that it has been created by querying the data dictionary.

```
CREATE SYNONYM dj_tracks2 FOR d_track_listings;
```

```
SELECT * FROM user_synonyms WHERE
```

```
table_NAME=UPPER('d_track_listings');
```

10. Drop the synonym that you created in question

```
DROP SYNONYM dj_tracks2;
```

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

OTHER DATABASE OBJECTS

EX.NO:14

DATE: 19 – 4 - 24

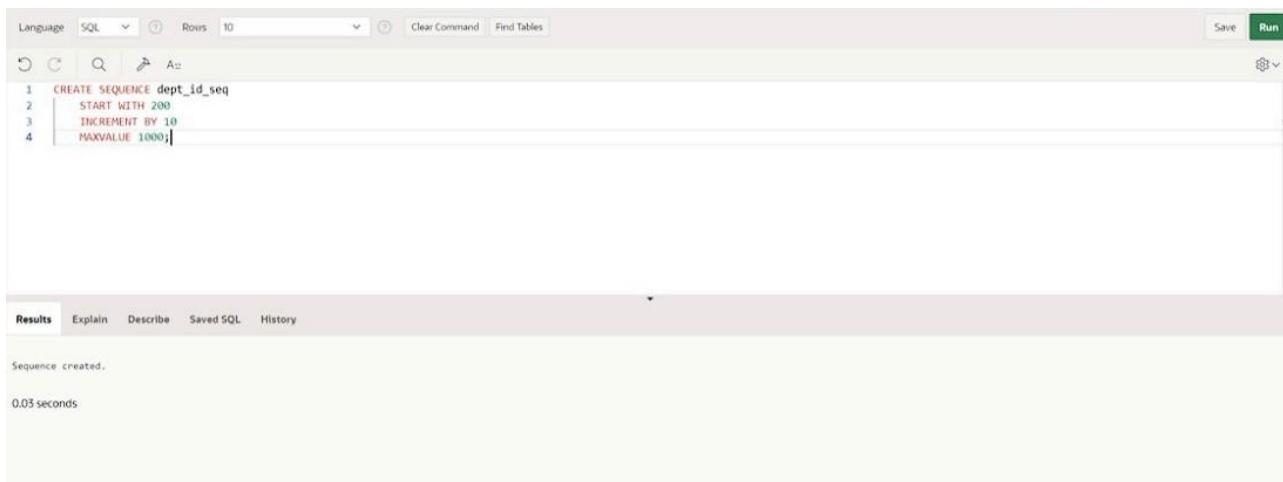
REG.NO: 220701239

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.

QUERY:

```
CREATE SEQUENCE dept_id_seq START WITH 200 INCREMENT BY 10 MAXVALUE 1000;
```

OUTPUT:



The screenshot shows a SQL query editor interface. The top bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and 'Run' buttons. The main area displays the following SQL code:

```
1 CREATE SEQUENCE dept_id_seq
2   START WITH 200
3   INCREMENT BY 10
4   MAXVALUE 1000;
```

Below the code, the 'Results' tab is selected, showing the output:

```
Sequence created.
0.03 seconds
```

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

QUERY:

```
SELECT sequence_name, max_value, increment_by, last_number FROM user_sequences;
```

OUTPUT:

The screenshot shows a SQL query editor interface. At the top, there are tabs for Language (set to SQL), Rows (set to 10), Clear Command, Find Tables, Save, and Run. The main area contains the following SQL code:

```
1 SELECT sequence_name, max_value, increment_by, last_number
2   FROM user_sequences;
3
```

Below the code, there is a results table with the following data:

SEQUENCE_NAME	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
DEPT_ID_SEQ	1000	10	200

At the bottom left of the results table, it says "1 rows returned in 0.02 seconds".

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.

QUERY:

```
INSERT INTO dept VALUES (dept_id_seq.nextval, 'Education');
```

OUTPUT:

SQL Scripts \ Results							
Script:	exercise14	Status:	Complete				
View:	<input type="radio"/> Detail	<input checked="" type="radio"/> Summary	<input type="radio"/>	Rows	15		
Number	Elapsed	Statement	Feedback	Rows			
1	0.03	INSERT INTO dept VALUES (dept_id_seq.nextval, 'Education')	1 row(s) inserted.	1			
Download							
row(s) 1 - 1 of 1							
1		1	0				
Statements Processed		Successful	With Errors				

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.

QUERY:

```
CREATE INDEX emp_dept_id_idx ON EMPLOYEES (department_id);
```

OUTPUT:

The screenshot shows a SQL query editor interface. The top bar includes 'Language' set to 'SQL', 'Rows' set to '10', and buttons for 'Save' and 'Run'. The main area contains the SQL command:

```
1 | CREATE INDEX emp_dept_id_idx ON EMPLOYEES (department_id);
2 |
```

Below the command, there is a results pane with tabs for 'Results', 'Explain', 'Describe', 'Saved SQL', and 'History'. The 'Results' tab is selected, displaying the output:

Index created.
0.03 seconds

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

QUERY:

```
SELECT index_name,table_name,uniqueness FROM user_indexes WHERE  
table_name='EMPLOYEES';
```

OUTPUT:

The screenshot shows a SQL query editor with the following details:

- Language: SQL
- Rows: 10
- Clear Command
- Find Tables
- Run button
- SQL code:

```
1 SELECT index_name, table_name, uniqueness  
2   FROM user_indexes  
3  WHERE table_name = 'EMPLOYEES';  
4
```
- Results tab selected
- Table structure:

INDEX_NAME	TABLE_NAME	UNIQUENESS
EMP_DEPT_ID_IDX	EMPLOYEES	NONUNIQUE
- 1 rows returned in 0.03 seconds
- Download link

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

CONTROLLING USER ACCESS

EX.NO:15

DATE: 25 - 4 - 24

REG.NO: 220701239

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

The CREATE SESSION system privilege

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

The CREATE TABLE privilege

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

You can, or anyone you have given those privileges to by using the WITH GRANT OPTION

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

Create a role containing the system privileges and grant the role to the users

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

The ALTER USER statement

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

Team 2 executes the GRANT statement. GRANT select ON departments TO <user1> ;

Team 1 executes the GRANT statement. GRANT select ON departments TO <user2> ;

7. Query all the rows in your DEPARTMENTS table.

```
SELECT * FROM departments;
```

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

Team 1 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name) VALUES (500, 'Education'); COMMIT;
```

Team 2 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name) VALUES (510, 'Administration'); COMMIT;
```

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

```
SELECT table_name FROM user_tables;
```

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

Team 1 revokes the privilege. REVOKE select ON departments FROM user2;

Team 2 revokes the privilege. REVOKE select ON departments FROM user1;

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

Team 1 executes this INSERT statement.

```
DELETE FROM departments WHERE department_id = 500; COMMIT;
```

Team 2 executes this INSERT statement.

```
DELETE FROM departments WHERE department_id = 510; COMMIT;
```

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

PL/SQL

CONTROL STRUCTURES

EX.NO:16

DATE: 26 – 4 - 24

REG.NO: 220701239

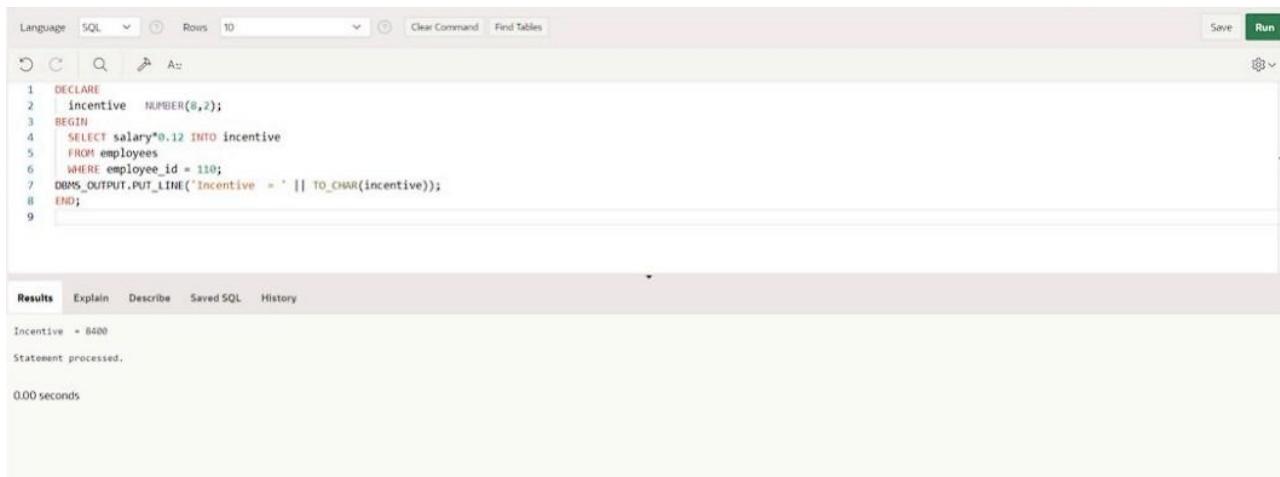
PROGRAM 1

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

QUERY:

```
DECLARE
    incentive NUMBER(8,2);
BEGIN
    SELECT salary*0.12 INTO incentive
    FROM employees
    WHERE employee_id = 110;
    DBMS_OUTPUT.PUT_LINE('Incentive = ' || TO_CHAR(incentive));
END;
```

OUTPUT:



The screenshot shows the Oracle SQL Developer interface. The top navigation bar includes 'Language' (set to SQL), 'Rows' (set to 10), 'Clear Command', 'Find Tables', 'Save', and 'Run'. Below the toolbar are standard database connection icons. The main workspace contains the PL/SQL code. The 'Results' tab at the bottom displays the output of the executed query.

```
1 DECLARE
2     incentive  NUMBER(8,2);
3 BEGIN
4     SELECT salary*0.12 INTO incentive
5     FROM employees
6     WHERE employee_id = 110;
7     DBMS_OUTPUT.PUT_LINE('Incentive = ' || TO_CHAR(incentive));
8 END;
```

Incentive = 8400
Statement processed.
0.00 seconds

PROGRAM 2

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

QUERY:

```
DECLARE
WELCOME varchar2(10) := 'welcome';
'welcome'; BEGIN
DBMS_Output.Put_Line("Welcome");
; END;
/
```

```
DECLARE
WELCOME varchar2(10) :=
BEGIN
DBMS_Output.Put_Line("Welcome")
END;
/
```

OUTPUT:

The screenshot shows a PL/SQL editor window in Oracle SQL Developer. The code entered is:

```
1 DECLARE
2   | WELCOME varchar2(10) := 'welcome';
3 BEGIN
4   DBMS_Output.Put_Line("Welcome");
5 END;
6 /
7
```

The results tab is selected, displaying the following error message in a yellow box:

```
Error at line 4/25: ORA-06550: line 4, column 25:
PLS-00201: identifier 'Welcome' must be declared
ORA-06512: at "SYS.100V_DBMS_SQL_APEX_230200", line 881
ORA-06550: line 4, column 3:
PL/SQL: Statement ignored
```

Below the error message, the original code is repeated:

```
2.   WELCOME varchar2(10) := 'welcome';
3. BEGIN
4.   DBMS_Output.Put_Line("Welcome");
5. END;
6. /
```

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

Sample table: employees

QUERY:

```
DECLARE
    salary_of_emp NUMBER(8,2);
    PROCEDURE approx_salary (
        emp      NUMBER,
        empsal IN OUT NUMBER,
        addless  NUMBER
    ) IS
    BEGIN
        empsal := empsal + addless;
    END;

BEGIN
    SELECT salary INTO salary_of_emp
    FROM employees
    WHERE employee_id = 122;
    DBMS_OUTPUT.PUT_LINE
    ('Before invoking procedure, salary_of_emp: ' || salary_of_emp);
    approx_salary (100, salary_of_emp, 1000);
    DBMS_OUTPUT.PUT_LINE
    ('After invoking procedure, salary_of_emp: ' || salary_of_emp);
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run, Undo, Redo, Find, Clear Command, Find Tables.
- Code Area:** PL/SQL code for a procedure named `approx_salary`. The code declares a variable `salary_of_emp` and performs a SELECT INTO operation to assign the value of `employee_id = 122` from the `employees` table to `salary_of_emp`.
- Results Tab:** Active tab, showing the output of the procedure execution.
- Output:**

```
Before invoking procedure, salary_of_emp: 6900
After invoking procedure, salary_of_emp: 7900
Statement processed.

0.00 seconds
```

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.

QUERY:

```
CREATE OR REPLACE PROCEDURE pri_bool(
    boo_name VARCHAR2,
    boo_val BOOLEAN
) IS
BEGIN
    IF boo_val IS NULL THEN
        DBMS_OUTPUT.PUT_LINE( boo_name || ' = NULL');
    ELSIF boo_val = TRUE THEN
        DBMS_OUTPUT.PUT_LINE( boo_name || ' = TRUE');
    ELSE
        DBMS_OUTPUT.PUT_LINE( boo_name || ' = FALSE');
    END IF;
END;
/
```

OUTPUT:

The screenshot shows a database query editor interface. At the top, there are tabs for Language (set to SQL), Rows (set to 10), and various command buttons like Save and Run. The main area contains the PL/SQL code for the procedure. Below the code, the Results tab is selected, showing the output: "Procedure created." and "0.03 seconds".

```
Language: SQL | Rows: 10 | Clear Command | Find Tables | Save | Run |     
```

```
1 CREATE OR REPLACE PROCEDURE pri_bool(
2     boo_name VARCHAR2,
3     boo_val BOOLEAN
4 ) IS
5 BEGIN
6     IF boo_val IS NULL THEN
7         DBMS_OUTPUT.PUT_LINE( boo_name || ' = NULL');
8     ELSIF boo_val = TRUE THEN
9         DBMS_OUTPUT.PUT_LINE( boo_name || ' = TRUE');
10    ELSE
11        DBMS_OUTPUT.PUT_LINE( boo_name || ' = FALSE');
12    END IF;
13 END;
14 /
15 |
```

Results Explain Describe Saved SQL History

Procedure created.
0.03 seconds

PROGRAM 5

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

QUERY:

```
DECLARE
  PROCEDURE pat_match (
    test_string  VARCHAR2,
    pattern      VARCHAR2
  ) IS
  BEGIN
    IF test_string LIKE pattern THEN
      DBMS_OUTPUT.PUT_LINE ('TRUE');
    ELSE
      DBMS_OUTPUT.PUT_LINE
      ('FALSE'); END IF;
    END;
  BEGIN
    pat_match('Blweate', 'B%a_e');
    pat_match('Blweate', 'B%A_E');
  END;
/
```

OUTPUT:



The screenshot shows the Oracle SQL developer interface. The code area contains the PL/SQL block provided above. The results tab shows the output of the procedure calls:

```
TRUE
FALSE
```

Below the results, a message states "Statement processed." and the execution time is listed as "0.00 seconds".

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.

QUERY:

DECLARE

```
num_small NUMBER := 8;
num_large NUMBER := 5;
num_temp NUMBER;
BEGIN

IF num_small > num_large THEN
    num_temp := num_small;
    num_small := num_large;
    num_large := num_temp;
END IF;
```

```
DBMS_OUTPUT.PUT_LINE ('num_small = '||num_small);
DBMS_OUTPUT.PUT_LINE ('num_large = '||num_large);
END;
/
```

OUTPUT:



```
2 num_small NUMBER := 8;
3 num_large NUMBER := 5;
4 num_temp NUMBER;
5 BEGIN
6
7 IF num_small > num_large THEN
8 num_temp := num_small;
9 num_small := num_large;
10 num_large := num_temp;
11 END IF;
12
13 DBMS_OUTPUT.PUT_LINE ('num_small = '||num_small);
14 DBMS_OUTPUT.PUT_LINE ('num_large = '||num_large);
15 END;
16 /
17
```

Results Explain Describe Saved SQL History

```
num_small = 5
num_large = 8

Statement processed.

0.00 seconds
```

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.

QUERY:

```
DECLARE
  PROCEDURE test1 (
    sal_achieve NUMBER,
    target_qty NUMBER,
    emp_id NUMBER
  )
  IS
    incentive NUMBER := 0;
    updated VARCHAR2(3) := 'No';
  BEGIN
    IF sal_achieve > (target_qty + 200) THEN
      incentive := (sal_achieve - target_qty)/4;
      UPDATE employees
      SET salary = salary + incentive
      WHERE employee_id = emp_id;
      updated := 'Yes';
    END IF;
    DBMS_OUTPUT.PUT_LINE (
      'Table updated? ' || updated || ',' ||
      'incentive = ' || incentive || '');
  );
END test1;
BEGIN
  test1(2300, 2000, 144);
  test1(3600, 3000,
145); END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with a code editor and a results tab.

```
1  DECLARE
2  PROCEDURE test1 (
3      sal_achieve NUMBER,
4      target_qty NUMBER,
5      emp_id NUMBER
6  )
7  IS
8      incentive NUMBER := 0;
9      updated VARCHAR2(10) := 'No';
10 BEGIN
11     IF sal_achieve > (target_qty + 200) THEN
12         incentive := (sal_achieve - target_qty)/4;
13     UPDATE employees
14     SET salary = salary + incentive
15     WHERE employee_id = emp_id;
16
17
```

Below the code, the results tab is selected, showing the following output:

Table updated? Yes, Incentive = 75.
Table updated? Yes, Incentive = 150.
1 row(s) updated.

0.02 seconds

PROGRAM 8

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

QUERY:

```
DECLARE
  PROCEDURE test1 (sal_achieve NUMBER)
  IS
    incentive NUMBER := 0;
  BEGIN
    IF sal_achieve > 44000 THEN
      incentive := 1800;
    ELSIF sal_achieve > 32000 THEN
      incentive := 800;
    ELSE
      incentive := 500;
    END IF;
    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE (
      'Sale achieved : ' || sal_achieve || ', incentive : ' || incentive || '');
  END test1;
BEGIN
  test1(45000);
  ;test1(36000);
  test1(28000);
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run
- Code (PL/SQL Procedure):**

```
1  DECLARE
2  PROCEDURE test1 (sal_achieve NUMBER)
3  IS
4      incentive NUMBER := 0;
5  BEGIN
6      IF sal_achieve > 44000 THEN
7          incentive := 1800;
8      ELSIF sal_achieve > 32000 THEN
9          incentive := 800;
10     ELSE
11         incentive := 500;
12     END IF;
13     DBMS_OUTPUT.NEW_LINE;
14     DBMS_OUTPUT.PUT_LINE (
15         'Sale achieved : ' || sal_achieve || ', incentive : ' || incentive || '.')
```
- Results Tab:** Shows the output of the procedure execution:

```
Sale achieved : 45000, incentive : 1800.
Sale achieved : 36000, incentive : 800.
Sale achieved : 28000, incentive : 500.
Statement processed.
```
- Timing:** 0.00 seconds

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.

QUERY:

```
SET SERVEROUTPUT ON
DECLARE
    tot_emp NUMBER;
    get_dep_id NUMBER;

BEGIN
    get_dep_id := 80;
    SELECT Count(*)
    INTO tot_emp
    FROM employees e
        join departments d
            ON e.department_id = d.department_id
    WHERE e.department_id = get_dep_id;
    dbms_output.Put_line ('The employees are in the department '||get_dep_id||' is: '
        ||To_char(tot_emp));
    IF tot_emp >= 45 THEN
        dbms_output.Put_line ('There are no vacancies in the department
        '||get_dep_id); ELSE
        dbms_output.Put_line ('There are '||to_char(45-tot_emp)||' vacancies in department
        '|| get_dep_id );
    END IF;
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Code Area:** PL/SQL script.

```
1 SET SERVEROUTPUT ON
2 DECLARE
3     tot_emp NUMBER;
4 BEGIN
5     SELECT Count(*)
6         INTO tot_emp
7         FROM employees e
8             JOIN departments d
9                 ON e.department_id = d.department_id
10                WHERE e.department_id = 50;
11
12    dbms_output.Put_line ('The employees are in the department 50: '
13                           ||to_char(tot_emp));
14
15    IF tot_emp >= 45 THEN
```
- Results Tab:** Shows the output of the script:

```
Sale achieved : 45000, incentive : 1800.
Sale achieved : 36000, incentive : 800.
Sale achieved : 28000, incentive : 500.
Statement processed.
```
- Timing:** 0.00 seconds.

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.

QUERY:

```
SET SERVEROUTPUT ON
DECLARE
    tot_emp NUMBER;
    get_dep_id NUMBER;

BEGIN
    get_dep_id := 80;
    SELECT Count(*)
    INTO tot_emp
    FROM employees e
        join departments d
            ON e.department_id = d.department_id
    WHERE e.department_id = get_dep_id;
    dbms_output.Put_line ('The employees are in the department '||get_dep_id||' is: '
        ||To_char(tot_emp));
    IF tot_emp >= 45 THEN
        dbms_output.Put_line ('There are no vacancies in the department
        '||get_dep_id); ELSE
        dbms_output.Put_line ('There are '||to_char(45-tot_emp)||' vacancies in department
        '|| get_dep_id );
    END IF;
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run
- Script Content (PL/SQL):**

```
1  DECLARE
2      tot_emp NUMBER;
3      get_dep_id NUMBER;
4
5  BEGIN
6      get_dep_id := 80;
7      SELECT Count(*)
8          INTO tot_emp
9         FROM employees e
10        JOIN departments d
11           ON e.department_id = d.dept_id
12      WHERE e.department_id = get_dep_id;
13
14      dbms_output.Put_line ('The employees are in the department'||get_dep_id||' is: '
15                             ||To_char(tot_emp));
```
- Results Tab:** The results show the output of the PL/SQL block:

```
The employees are in the department 80 is: 4
There are 41 vacancies in department 80
Statement processed.
```
- Timing:** 0.03 seconds

PROGRAM 11

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

QUERY:

```
DECLARE
  v_employee_id employees.employee_id%TYPE;
  v_full_name employees.first_name%TYPE;
  v_job_id employees.job_id%TYPE;
  v_hire_date
    employees.hire_date%TYPE; v_salary
    employees.salary%TYPE; CURSOR
  c_employees IS
    SELECT employee_id, first_name || ' ' || last_name AS full_name, job_id, hire_date,
salary
      FROM employees;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Employee ID | Full Name | Job Title | Hire Date |
Salary'); DBMS_OUTPUT.PUT_LINE('_____');
  OPEN c_employees;
  FETCH c_employees INTO v_employee_id, v_full_name, v_job_id, v_hire_date,
v_salary; WHILE c_employees%FOUND LOOP
  DBMS_OUTPUT.PUT_LINE(v_employee_id || ' ' || v_full_name || ' ' || v_job_id
|| ' ' || v_hire_date || ' ' || v_salary);
  FETCH c_employees INTO v_employee_id, v_full_name, v_job_id, v_hire_date,
v_salary; END LOOP;
  CLOSE c_employees;
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run
- Text Area (PL/SQL):**

```
1  DECLARE
2    v_employee_id employees.employee_id%TYPE;
3    v_full_name employees.first_name%TYPE;
4    v_job_id employees.job_id%TYPE;
5    v_hire_date employees.hire_date%TYPE;
6    v_salary employees.salary%TYPE;
7    CURSOR c_employees IS
8      SELECT employee_id, first_name || ' ' || last_name AS full_name, job_id, hire_date, salary
9        FROM employees;
10   BEGIN
11     DBMS_OUTPUT.PUT_LINE('Employee ID | Full Name | Job Title | Hire Date | Salary');
12     DBMS_OUTPUT.PUT_LINE('-----');
13     OPEN c_employees;
```
- Results Tab:** Employee ID | Full Name | Job Title | Hire Date | Salary
- Data Output:**

Employee ID	Full Name	Job Title	Hire Date	Salary
110	John Smith	sales_rep	02/28/1995	70000
115	Emily Johnson	hr_rep	01/12/1998	5000
122	Jaunty Janu	ac_account	01/01/2004	6000
114	Den Davies	st_clerk	02/03/1999	11800
142	Jane Doe	ac_account	03/05/1997	30000
115	Vijaya Nohan	st_clerk	02/22/1998	4000
- Message:** Statement processed.

PROGRAM 12

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

QUERY:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT e.employee_id, e.first_name, m.first_name AS manager_name
    FROM employees e
    LEFT JOIN employees m ON e.manager_id = m.employee_id;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  FETCH emp_cursor INTO emp_record;
  WHILE emp_cursor%FOUND LOOP
    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_record.employee_id);
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.first_name);
    DBMS_OUTPUT.PUT_LINE('Manager Name: ' ||
      emp_record.manager_name); DBMS_OUTPUT.PUT_LINE('');
    FETCH emp_cursor INTO emp_record;
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

OUTPUT:

The screenshot shows a database query editor interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Code Area:** PL/SQL block. The code uses a cursor to select employee and manager names from the employees table, then loops through the cursor to print each employee's ID, name, and manager's name.

```
1 DECLARE
2   CURSOR emp_cursor IS
3     SELECT e.employee_id, e.first_name, m.first_name AS manager_name
4       FROM employees e
5      LEFT JOIN employees m ON e.manager_id = m.employee_id;
6   emp_record emp_cursor%ROWTYPE;
7 BEGIN
8   OPEN emp_cursor;
9   FETCH emp_cursor INTO emp_record;
10  WHILE emp_cursor%FOUND LOOP
11    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_record.employee_id);
12    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_record.first_name);
13    DBMS_OUTPUT.PUT_LINE('Manager Name: ' || emp_record.manager_name);
14  END LOOP;
15 END;
```

- Results Area:** Displays the output of the PL/SQL block. It shows three records:

Employee ID	Employee Name	Manager Name
142	Jane	
110	John	
125	Emily	

PROGRAM 13

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

QUERY:

```
DECLARE
    CURSOR job_cursor IS
        SELECT e.job_id, j.lowest_sal
        FROM job_grade j,employees e;
    job_record
    job_cursor%ROWTYPE; BEGIN
        OPEN job_cursor;
        FETCH job_cursor INTO job_record;
        WHILE job_cursor%FOUND LOOP
            DBMS_OUTPUT.PUT_LINE('Job ID: ' || job_record.job_id);
            DBMS_OUTPUT.PUT_LINE('Minimum Salary: ' ||
                job_record.lowest_sal); DBMS_OUTPUT.PUT_LINE('');
            FETCH job_cursor INTO job_record;
        END LOOP;
        CLOSE job_cursor;
    END;
/
```

OUTPUT:

The screenshot shows a SQL query editor interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Code Area:** A PL/SQL block is displayed:

```
1  DECLARE
2      CURSOR job_cursor IS
3          SELECT e.job_id, j.lowest_sal
4          FROM job_grade j,employees e;
5      job_record job_cursor%ROWTYPE;
6  BEGIN
7      OPEN job_cursor;
8      FETCH job_cursor INTO job_record;
9      WHILE job_cursor%FOUND LOOP
10         DBMS_OUTPUT.PUT_LINE('Job ID: ' || job_record.job_id);
11         DBMS_OUTPUT.PUT_LINE('Minimum Salary: ' || job_record.lowest_sal);
12         DBMS_OUTPUT.PUT_LINE('-----');
13     FETCH job_cursor INTO job_record;
```
- Results Area:** The results tab is selected, showing the output of the PL/SQL block:

```
Job ID: sales_rep
Minimum Salary: 2500
-----
Job ID: hr_rep
Minimum Salary: 2500
-----
Job ID: ac_account
Minimum Salary: 2500
-----
Job ID: st_clerk
Minimum Salary: 2500
```

PROGRAM 15

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

QUERY:

```
DECLARE
  CURSOR employees_cur IS
    SELECT employee_id, last_name,
           job_id, start_date FROM employees NATURAL join
           job_history; emp_start_date DATE;
BEGIN
  dbms_output.Put_line(Rpad('Employee ID', 15) || Rpad('Last Name', 25) || Rpad('Job
Id', 35) || 'Start Date');
  dbms_output.Put_line('_____');
FOR emp_sal_rec IN employees_cur LOOP
  -- find out most recent end_date in job_history
  SELECT Max(end_date) + 1
  INTO emp_start_date
  FROM job_history
  WHERE employee_id = emp_sal_rec.employee_id;
  IF emp_start_date IS NULL THEN
    emp_start_date := emp_sal_rec.start_date;
  END IF;
  dbms_output.Put_line(Rpad(emp_sal_rec.employee_id, 15)
    || Rpad(emp_sal_rec.last_name, 25)
    || Rpad(emp_sal_rec.job_id, 35)
    || To_char(emp_start_date, 'dd-mon-yyyy'));
END LOOP;
END;
/
```

OUTPUT:

The screenshot shows a SQL developer interface. The code area contains a PL/SQL block that declares a cursor, selects data from the employees and job_history tables, and then prints the results using dbms_output.Put_line. The results panel shows two rows of data: Employee ID 125, Last Name Johnson, Job Id hr_rep, and Start Date 22-apr-1999.

```
1  DECLARE
2  CURSOR employees_cur IS
3    SELECT employee_id,
4           last_name,
5           job_id,
6           start_date
7    FROM employees
8    NATURAL JOIN job_history;
9    emp_start_date DATE;
10 BEGIN
11   dbms_output.Put_line(Rpad('Employee ID', 15)
12                         ||Rpad('Last Name', 25)
13                         ||Rpad('Job Id', 25)
14                         ||'Start Date');
15
16  dbms_output.Put_line('-----');
17
```

Employee ID	Last Name	Job Id	Start Date
125	Johnson	hr_rep	22-apr-1999
125	Johnson	hr_rep	22-apr-1999

Statement processed.

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

PROCEDURES AND FUNCTIONS

EX.NO:17

DATE: 02 – 5 - 24

REG.NO: 220701239

Program 1

FACTORIAL OF A NUMBER USING FUNCTION

QUERY:

```
DECLARE
    fac NUMBER := 1;
    n NUMBER := :1;
BEGIN
    WHILE n > 0 LOOP
        fac := n * fac;
        n := n - 1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(fac)
    ;
END;
```

OUTPUT:

The screenshot shows a PL/SQL editor window in Oracle SQL Developer. The code is a factorial calculation using a WHILE loop:

```
1 DECLARE
2     fac NUMBER := 1;
3     n NUMBER := 1; -- Use a bind variable instead of "%i"
4 BEGIN
5     WHILE n > 0 LOOP
6         fac := n * fac;
7         n := n - 1;
8     END LOOP;
9     DBMS_OUTPUT.PUT_LINE(fac);
10 END;
```

Below the code, the results tab is selected, showing the output of the statement:

120
Statement processed.
0.00 seconds

Program 2

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

QUERY:

```
CREATE OR REPLACE PROCEDURE get_book_info (
    p_book_id IN NUMBER,
    p_title IN OUT VARCHAR2,
    p_author OUT VARCHAR2,
    p_year_published OUT NUMBER
)
AS
BEGIN
    SELECT title, author, year_published INTO p_title, p_author, p_year_published
    FROM books
    WHERE book_id = p_book_id;

    p_title := p_title || ' - Retrieved';
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_title := NULL;
        p_author := NULL;
        p_year_published :=
NULL; END;
```

```

DECLARE
    v_book_id NUMBER := 1;
    v_title VARCHAR2(100);
    v_author VARCHAR2(100);
    v_year_published NUMBER;
BEGIN
    v_title := 'Initial Title';

    get_book_info(p_book_id => v_book_id, p_title => v_title, p_author => v_author,
    p_year_published => v_year_published);

    DBMS_OUTPUT.PUT_LINE('Title: ' || v_title);
    DBMS_OUTPUT.PUT_LINE('Author: ' || v_author);
    DBMS_OUTPUT.PUT_LINE('Year Published: ' || v_year_published);
END;
/

```

OUTPUT:

The screenshot shows the Oracle SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Clear Command** and **Find Tables** buttons
- Run** button
- Code Area:**

```

1 DECLARE
2     v_book_id NUMBER := 1;
3     v_title VARCHAR2(100);
4     v_author VARCHAR2(100);
5     v_year_published NUMBER;
6 BEGIN
7     v_title := 'Initial Title';
8
9     get_book_info(p_book_id => v_book_id, p_title => v_title, p_author => v_author, p_year_published => v_year_published);
10
11    DBMS_OUTPUT.PUT_LINE('Title: ' || v_title);
12    DBMS_OUTPUT.PUT_LINE('Author: ' || v_author);
13    DBMS_OUTPUT.PUT_LINE('Year Published: ' || v_year_published);
14 END;

```
- Results Tab:**
 - Output rows:
 - Title: To Kill a Mockingbird - Retrieved
 - Author: Harper Lee
 - Year Published: 1960
 - Note: Statement processed.
 - Time: 0.01 seconds

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

TRIGGER

EX.NO:18

DATE: 09 – 5 - 24

REG.NO: 220701239

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.

QUERY:

```
CREATE OR REPLACE TRIGGER prevent_parent_deletion
BEFORE DELETE ON parent_table
FOR EACH
ROW DECLARE
    child_exists EXCEPTION;
    PRAGMA EXCEPTION_INIT(child_exists, -
    20001); v_child_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_child_count FROM child_table WHERE parent_id =
:OLD.parent_id;
    IF v_child_count > 0 THEN
        RAISE child_exists;
    END IF;
EXCEPTION
    WHEN child_exists THEN
        RAISE_APPLICATION_ERROR(-20001, 'Cannot delete parent record while child
records exist.');
END;
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run, Undo, Redo, Find, Clear Command, Find Tables.
- Code:** The code creates a trigger named "prevent_parent_deletion" that prevents the deletion of a parent record if it has children. It uses a cursor to count children and raises an application error if any exist.

```
1 CREATE OR REPLACE TRIGGER prevent_parent_deletion
2 BEFORE DELETE ON parent_table
3 FOR EACH ROW
4 DECLARE
5   child_exists EXCEPTION;
6   PRAGMA EXCEPTION_INIT(child_exists, -20001);
7   v_child_count NUMBER;
8 BEGIN
9   SELECT COUNT(*) INTO v_child_count FROM child_table WHERE parent_id = :OLD.parent_id;
10  IF v_child_count > 0 THEN
11    RAISE child_exists;
12  END IF;
13 EXCEPTION
14 WHEN child_exists THEN
15   RAISE_APPLICATION_ERROR(-20001, 'Cannot delete parent record while child records exist.');
16 END;
17 /
```

- Results Tab:** Shows the message "Trigger created." and a execution time of "0.04 seconds".

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.

QUERY:

```
CREATE OR REPLACE TRIGGER check_duplicates
BEFORE INSERT OR UPDATE ON unique_values_table
FOR EACH ROW
DECLARE
    duplicate_found EXCEPTION;
    PRAGMA EXCEPTION_INIT(duplicate_found, -
                           20002);
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM
        unique_values_table WHERE unique_col =
        :NEW.unique_col AND id != :NEW.id; IF v_count > 0 THEN
        RAISE
        duplicate_found; END IF;
EXCEPTION
    WHEN duplicate_found THEN
        RAISE_APPLICATION_ERROR(-20002, 'Duplicate value found in unique_col.');
END;
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Code Area:** Contains PL/SQL code for creating a trigger. The code declares an exception 'duplicate_found', initializes it with -20002, and performs a SELECT COUNT(*) into 'v_count' from 'unique_values_table' where 'unique_col' = :NEW.unique_col AND id != :NEW.id. If 'v_count' is greater than 0, it raises the 'duplicate_found' exception. An exception block handles this raise, raising an application error with message 'Duplicate value found in unique_col.' with error code -20002.
- Status Bar:** Shows 'Trigger created.', '0.04 seconds' execution time, and tabs for Results, Explain, Describe, Saved SQL, and History.

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.

QUERY:

```
CREATE OR REPLACE TRIGGER check_threshold
BEFORE INSERT OR UPDATE ON threshold_table
FOR EACH ROW
DECLARE
    threshold_exceeded EXCEPTION;
    PRAGMA EXCEPTION_INIT(threshold_exceeded, -20003);
    v_sum NUMBER;
    v_threshold NUMBER := 10000; -- Set your threshold here
BEGIN
    SELECT SUM(value_col) INTO v_sum FROM threshold_table;
    v_sum := v_sum + :NEW.value_col;
    IF v_sum > v_threshold THEN
        RAISE threshold_exceeded;
    END IF;
EXCEPTION
    WHEN threshold_exceeded THEN
        RAISE_APPLICATION_ERROR(-20003, 'Threshold exceeded for value_col.');
END;
```

OUTPUT:

The screenshot shows a SQL editor interface with the following details:

- Language:** SQL
- Rows:** 10
- Toolbar:** Save, Run
- Code:** The code creates a trigger named "check_threshold" that runs before insert or update on the "threshold_table". It declares variables "v_sum" and "v_threshold", initializes "v_threshold" to 10000, and then performs a SELECT operation to sum all values in the "value_col" of the "threshold_table" into "v_sum". It then adds the new value from the ":NEW.value_col" to "v_sum". If "v_sum" exceeds "v_threshold", it raises the "threshold_exceeded" exception. An exception block handles this by raising an application error with message "Threshold exceeded for value_col.".
- Results:** Trigger created.
- Timing:** 0.04 seconds

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.

QUERY:

```
CREATE OR REPLACE TRIGGER log_changes
AFTER UPDATE ON main_table
FOR EACH
ROW BEGIN
    INSERT INTO audit_table (audit_id, changed_id, old_col1, new_col1, old_col2,
    new_col2, change_time)
    VALUES (audit_seq.NEXTVAL, :OLD.id, :OLD.col1, :NEW.col1, :OLD.col2, :NEW.col2,
    SYSTIMESTAMP);
END;
```

OUTPUT:



The screenshot shows a SQL developer interface with the following details:

- Toolbar:** Language (SQL), Rows (10), Clear Command, Find Tables, Save, Run.
- Query Editor:** Contains the PL/SQL code for creating the trigger.
- Results Tab:** Shows the output of the trigger creation command.
- Output:**
 - Trigger created.
 - 0.03 seconds

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.

QUERY:

```
CREATE OR REPLACE TRIGGER log_user_activity
AFTER INSERT OR UPDATE OR DELETE ON activity_table
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO user_activity_log (log_id, action, table_name, record_id,
change_time)
        VALUES (activity_log_seq.NEXTVAL, 'INSERT', 'activity_table', :NEW.id,
SYSTIMESTAMP);
    ELSIF UPDATING THEN
        INSERT INTO user_activity_log (log_id, action, table_name, record_id,
change_time)
        VALUES (activity_log_seq.NEXTVAL, 'UPDATE', 'activity_table',
:NEW.id, SYSTIMESTAMP);
    ELSIF DELETING THEN
        INSERT INTO user_activity_log (log_id, action, table_name, record_id,
change_time)
        VALUES (activity_log_seq.NEXTVAL, 'DELETE', 'activity_table', :OLD.id,
SYSTIMESTAMP);
    END IF;
END;
```

OUTPUT:

The screenshot shows a MySQL Workbench interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run
- Text Area Content:**

```
1 CREATE OR REPLACE TRIGGER log_user_activity
2 AFTER INSERT OR UPDATE OR DELETE ON activity_table
3 FOR EACH ROW
4 BEGIN
5     IF INSERTING THEN
6         INSERT INTO user_activity_log (log_id, action, table_name, record_id, change_time)
7             VALUES (activity_log_seq.NEXTVAL, 'INSERT', 'activity_table', :NEW.id, SYSTIMESTAMP);
8     ELSIF UPDATING THEN
9         INSERT INTO user_activity_log (log_id, action, table_name, record_id, change_time)
10            VALUES (activity_log_seq.NEXTVAL, 'UPDATE', 'activity_table', :NEW.id, SYSTIMESTAMP);
11    ELSIF DELETING THEN
12        INSERT INTO user_activity_log (log_id, action, table_name, record_id, change_time)
13            VALUES (activity_log_seq.NEXTVAL, 'DELETE', 'activity_table', :OLD.id, SYSTIMESTAMP);
```
- Results Tab:** Trigger created.
- Timing:** 0.03 seconds

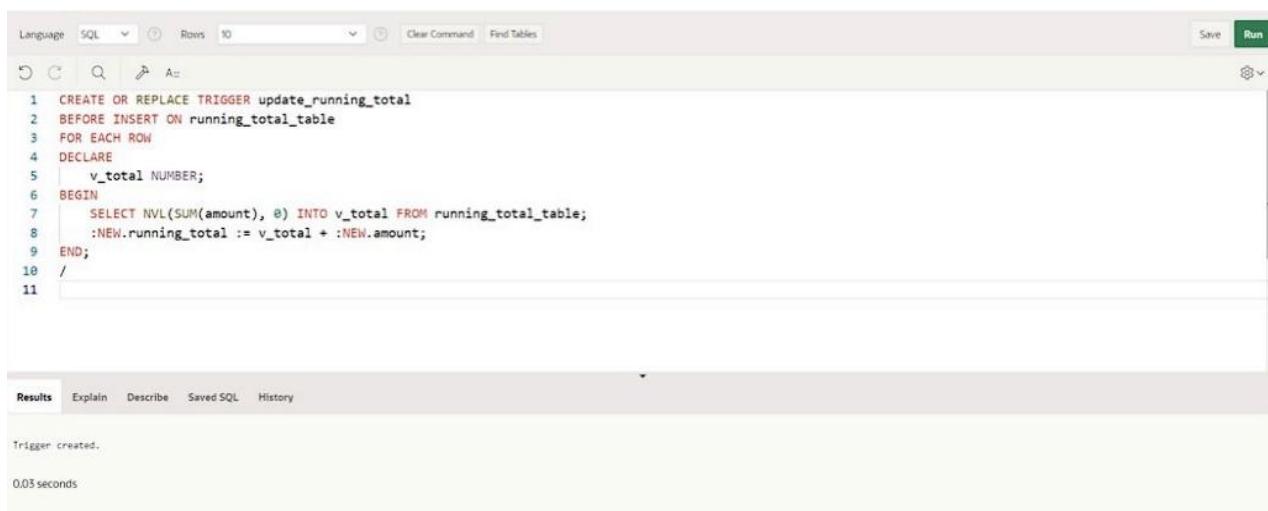
Program 6

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.

QUERY:

```
CREATE OR REPLACE TRIGGER update_running_total
BEFORE INSERT ON running_total_table
FOR EACH
ROW DECLARE
    v_total NUMBER;
BEGIN
    SELECT NVL(SUM(amount), 0) INTO v_total FROM running_total_table;
    :NEW.running_total := v_total + :NEW.amount;
END;
```

OUTPUT:



The screenshot shows the Oracle SQL Developer interface with a SQL command window. The command window contains the PL/SQL code for creating a trigger named 'update_running_total'. The code uses a before insert trigger on the 'running_total_table' for each row. It declares a variable 'v_total' of type NUMBER. Inside the trigger body, it selects the sum of 'amount' from the table into 'v_total', and then updates the ':NEW.running_total' to be the current value plus the new value of 'amount'. The code ends with a slash to terminate the trigger definition. Below the command window, the results tab shows the output: 'Trigger created.' and '0.03 seconds'.

```
Language: SQL | Rows: 10 | Clear Command | Find Tables | Save | Run |   
```

```
1 CREATE OR REPLACE TRIGGER update_running_total
2 BEFORE INSERT ON running_total_table
3 FOR EACH ROW
4 DECLARE
5     v_total NUMBER;
6 BEGIN
7     SELECT NVL(SUM(amount), 0) INTO v_total FROM running_total_table;
8     :NEW.running_total := v_total + :NEW.amount;
9 END;
10 /
11 
```

Results	Explain	Describe	Saved SQL	History
---------	---------	----------	-----------	---------

Trigger created.
0.03 seconds

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

QUERY:

```
CREATE OR REPLACE TRIGGER validate_order
BEFORE INSERT ON orders
FOR EACH
ROW DECLARE
    v_stock NUMBER;
    insufficient_stock EXCEPTION;
    PRAGMA EXCEPTION_INIT(insufficient_stock, -
20004); BEGIN
    SELECT stock_quantity INTO v_stock FROM items WHERE item_id =
:NEW.item_id; IF v_stock < :NEW.order_quantity THEN
        RAISE
    insufficient_stock; END IF;
    UPDATE items SET stock_quantity = stock_quantity - :NEW.order_quantity
WHERE item_id = :NEW.item_id;
EXCEPTION
    WHEN insufficient_stock THEN
        RAISE_APPLICATION_ERROR(-20004, 'Insufficient stock for the item.');
END;
```

OUTPUT:

The screenshot shows a SQL developer interface with the following details:

- Language:** SQL
- Rows:** 10
- Buttons:** Save, Run, Clear Command, Find Tables
- Code:**

```
1 CREATE OR REPLACE TRIGGER validate_order
2 BEFORE INSERT ON orders
3 FOR EACH ROW
4 DECLARE
5     v_stock NUMBER;
6     insufficient_stock EXCEPTION;
7     PRAGMA EXCEPTION_INIT(insufficient_stock, -20004);
8 BEGIN
9     SELECT stock_quantity INTO v_stock FROM items WHERE item_id = :NEW.item_id;
10    IF v_stock < :NEW.order_quantity THEN
11        RAISE insufficient_stock;
12    END IF;
13    UPDATE items SET stock_quantity = stock_quantity - :NEW.order_quantity WHERE item_id = :NEW.item_id;
14 EXCEPTION
15 WHEN insufficient_stock THEN
```
- Results:** Trigger created.
- Time:** 0.05 seconds

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

MONGO DB

EX.NO:19

DATE: 16 - 5 - 24

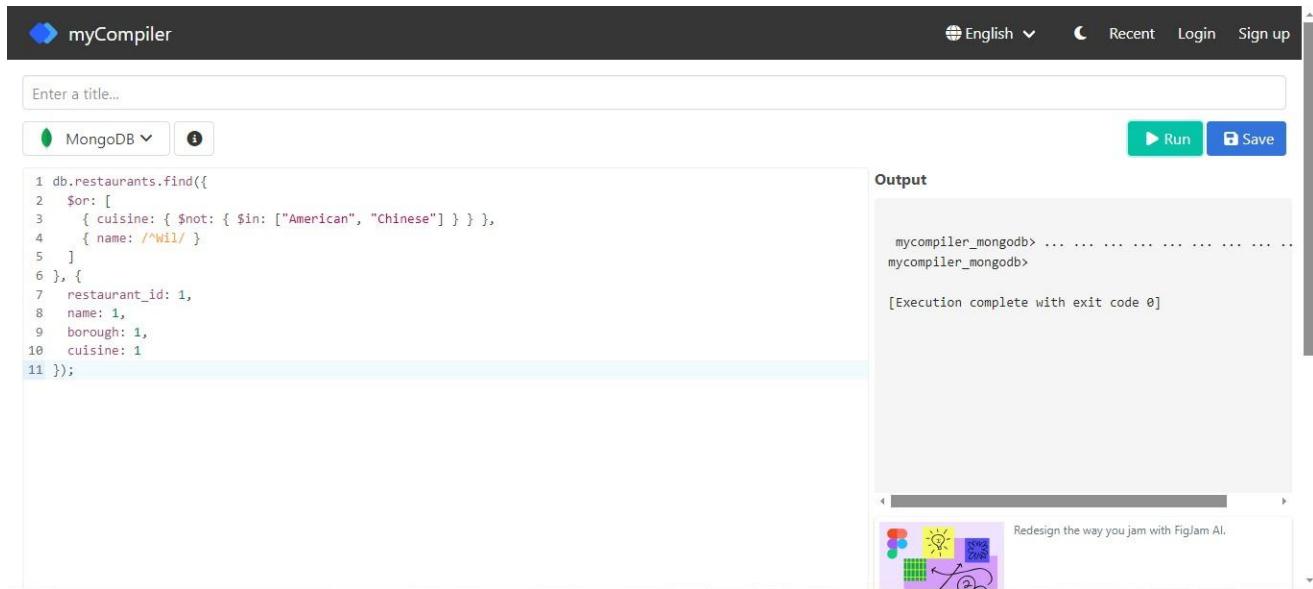
REG.NO: 220701239

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'.

QUERY:

```
db.restaurants.find( { $or: [ { name: /^Wil/ }, { cuisine: { $nin: ['American', 'Chinese'] } } ], { restaurant_id: 1, name: 1, borough: 1, cuisine: 1 } );
```

OUTPUT:



The screenshot shows the myCompiler web application interface. At the top, there is a navigation bar with a logo, language selection (English), recent projects, login, and sign-up links. Below the header, there is a search bar labeled "Enter a title...". On the left, a code editor window displays the following MongoDB query:

```
1 db.restaurants.find({  
2   $or: [  
3     { cuisine: { $not: { $in: ["American", "Chinese"] } } },  
4     { name: /^wil/ }  
5   ]  
6 }, {  
7   restaurant_id: 1,  
8   name: 1,  
9   borough: 1,  
10  cuisine: 1  
11 });
```

On the right, there is an "Output" window showing the command-line interface output:

```
mycompiler_mongodb> ...  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

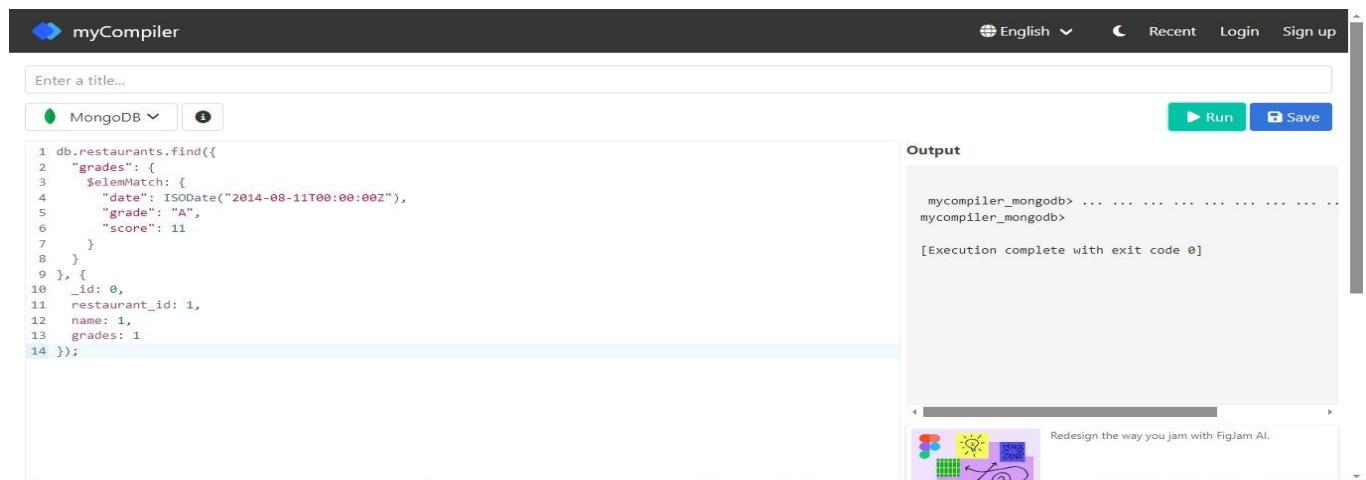
At the bottom of the interface, there is a watermark for FigJam AI.

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.

QUERY:

```
db.restaurants.find( { grades: { $elemMatch: { grade: "A",score: 11, date: ISODate("2014-08-11T00:00:00Z") } } }, { restaurant_id: 1, name: 1,grades: 1 } );
```

OUTPUT:



The screenshot shows the myCompiler MongoDB interface. The left panel contains the MongoDB query:

```
1 db.restaurants.find({  
2   "grades": {  
3     $elemMatch: {  
4       "date": ISODate("2014-08-11T00:00:00Z"),  
5       "grade": "A",  
6       "score": 11  
7     }  
8   }  
9 }, {  
10   _id: 0,  
11   restaurant_id: 1,  
12   name: 1,  
13   grades: 1  
14 });
```

The right panel shows the "Output" window with the command prompt "mycompiler_mongodb> ...". It displays the command run and the message "[Execution complete with exit code 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".

QUERY:

```
db.restaurants.find( { "grades.1.grade": "A", "grades.1.score": 9,
"grades.1.date": ISODate("2014-08-11T00:00:00Z") }, { restaurant_id: 1, name:
1, grades: 1 } );
```

OUTPUT:

The screenshot shows a MongoDB shell interface. On the left, the code block is displayed:

```
1 db.restaurants.find({
2   "grades.1.grade": "A",
3   "grades.1.score": 9,
4   "grades.1.date": ISODate("2014-08-11T00:00:00Z")
5 }, {
6   _id: 0,
7   restaurant_id: 1,
8   name: 1,
9   grades: 1
10});
```

On the right, the 'Output' panel shows the results of the query execution:

```
mycompiler_mongodb> ...
mycompiler_mongodb>
[Execution complete with exit code 0]
```

The system tray at the bottom of the screen indicates the date as 5/24/2024, the time as 11:41 PM, and the temperature as 31°C.

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

QUERY:

```
db.restaurants.find({$and : [{"address.coord.1": {$gt : 42}}, {"address.coord.1": {$lte : 52}}]}, {_id:0, restaurant_id:1, name:1, address:1})
```

OUTPUT:

The screenshot shows a MongoDB shell interface. On the left, the query is entered:

```
1 db.restaurants.find({  
2   "address.coord.1": { $gt: 42, $lte: 52 }  
3 }, {  
4   _id: 0,  
5   restaurant_id: 1,  
6   name: 1,  
7   address: 1  
8 })$
```

On the right, the 'Output' panel shows the results of the query execution:

```
mycompiler_mongodb> ...  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

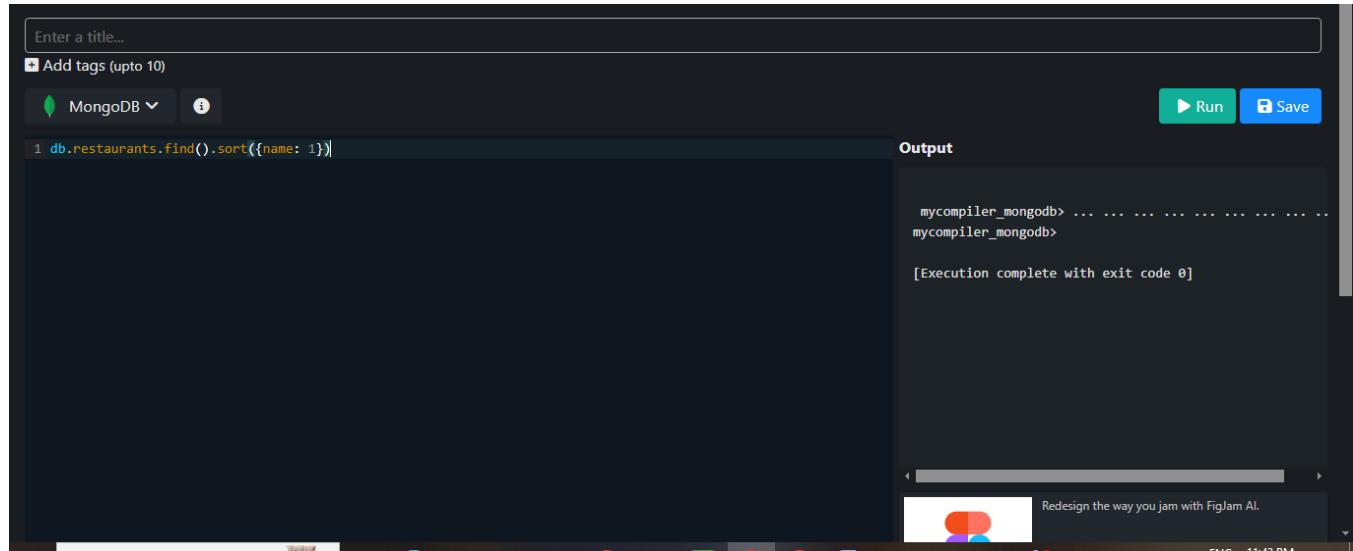
A small watermark at the bottom right of the interface reads: "Redesign the way you jam with FigJam AI."

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

QUERY:

```
db.restaurants.find({}, { _id: 0 }).sort({ name: 1 });
```

OUTPUT:



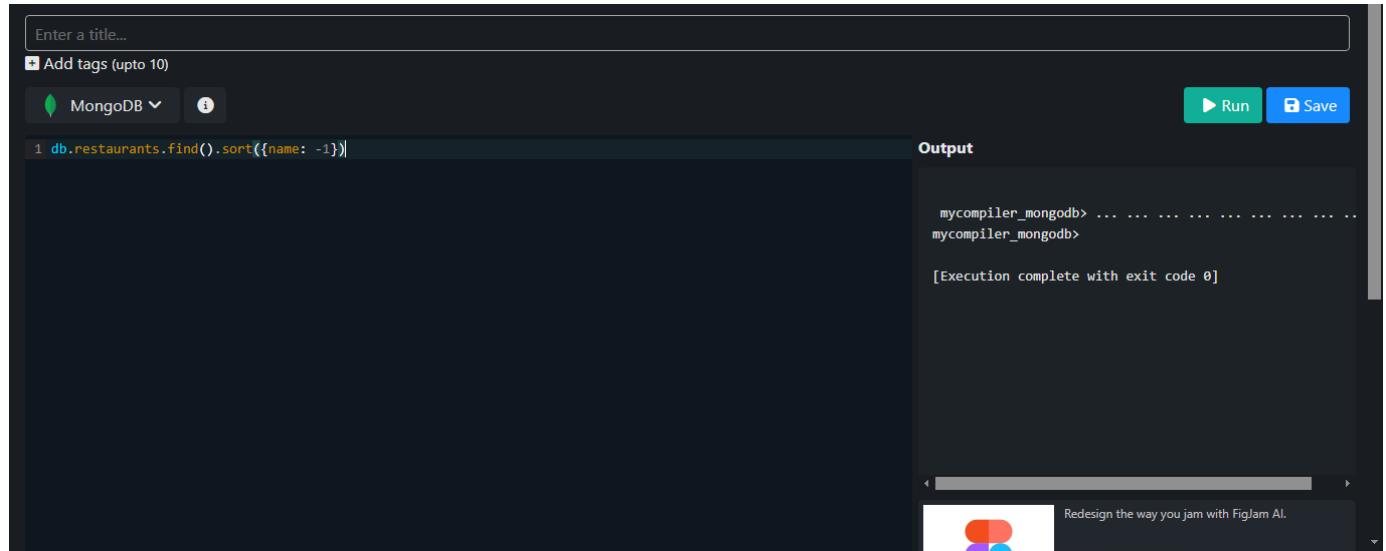
The screenshot shows a MongoDB shell interface. The top bar includes fields for 'Enter a title...', 'Add tags (upto 10)', a dropdown for 'MongoDB', and buttons for 'Run' and 'Save'. Below this, the code input field contains the command: `1 db.restaurants.find().sort({name: 1})`. To the right, the 'Output' panel displays the results of the command execution. The output shows the prompt 'mycompiler_mongodb>' followed by several dots indicating continuation. It also includes the message '[Execution complete with exit code 0]'. A small advertisement for FigJam AI is visible at the bottom of the output panel.

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

QUERY:

```
db.restaurants.find({ }, { _id: 0 }).sort({ name: -1 })
```

OUTPUT:



The screenshot shows a MongoDB shell interface. On the left, there is a code editor window with the following content:

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

On the right, there is an "Output" window displaying the results of the query. The output shows the command entered and the response from the MongoDB shell:

```
mycompiler_mongodb> ... . . . . . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

The interface includes a title bar with "Enter a title...", a "Add tags (upto 10)" button, and a toolbar with "MongoDB" dropdown, "Run" button, and "Save" button.

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

QUERY:

```
db.restaurants.find({ }, { _id: 0 }).sort({ cuisine: 1, borough: -1 })
```

OUTPUT:

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

QUERY:

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

OUTPUT:

The screenshot shows a MongoDB query editor interface. On the left, there is a code editor with the following query:

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

On the right, there is an "Output" panel displaying the results of the query execution:

```
mycompiler_mongodb> ... . . . . . . . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

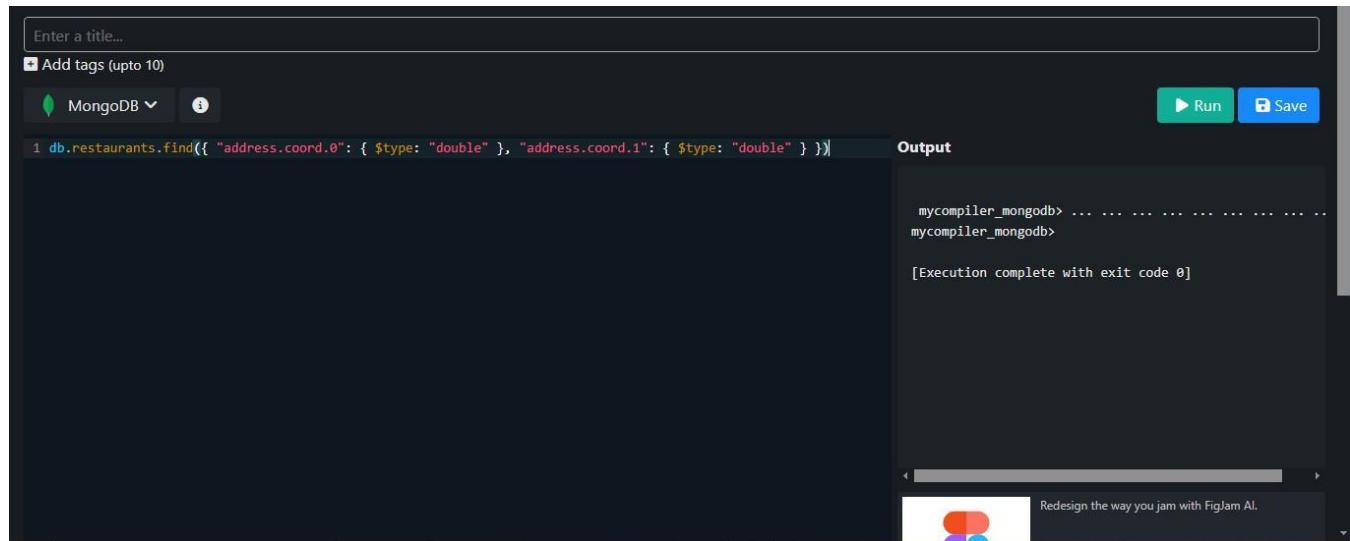
At the bottom right of the interface, there is a small advertisement for FigJam AI.

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

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB query editor interface. On the left, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below these are two dropdown menus: one for "MongoDB" and another with a question mark icon. On the right, there are two buttons: "Run" (green) and "Save" (blue). The main area contains a code input field with the following query:

```
1 db.restaurants.find({ "address.coord.0": { $type: "double" }, "address.coord.1": { $type: "double" } })
```

Below the code input is a "Output" panel. It displays the command "mycompiler_mongodb>" followed by the response "mycompiler_mongodb>". At the bottom of the output panel, it says "[Execution complete with exit code 0]".

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.

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. On the left, there is a text input field labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below these are two buttons: "MongoDB" with a dropdown arrow and a small info icon. To the right are "Run" and "Save" buttons. The main area contains a command line with the following text:

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

On the right side, there is a "Output" panel with the following text:

```
mycompiler_mongodb> ... . . . . . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

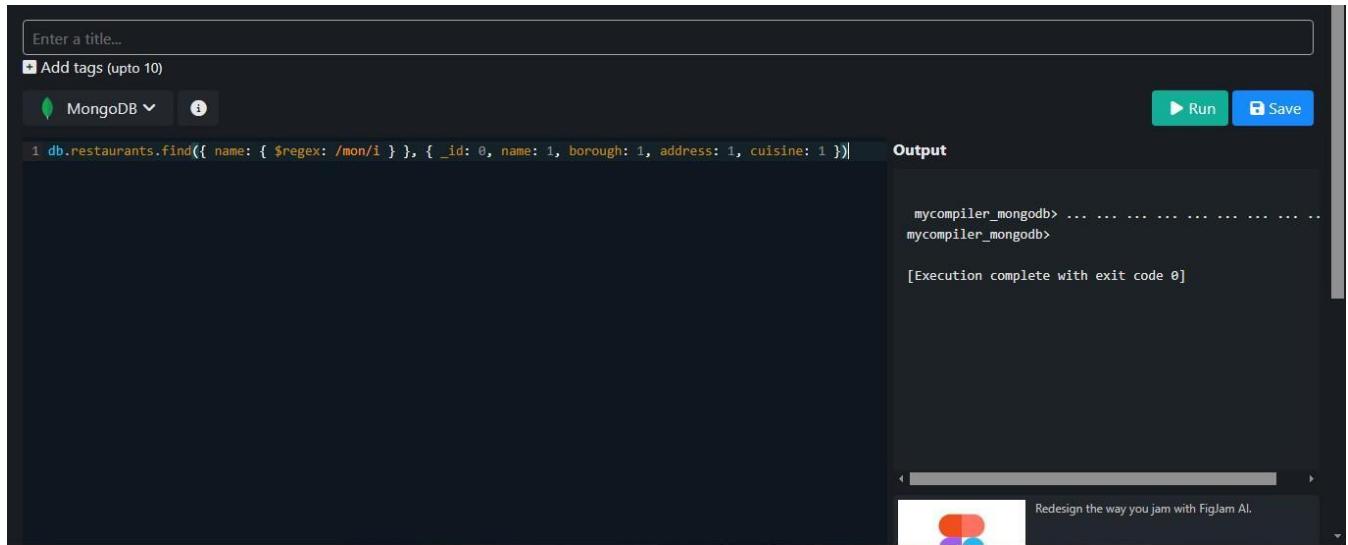
At the bottom right of the interface, there is a watermark for "Redesign the way you jam with FigJam AI." followed by the FigJam logo.

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.

QUERY:

```
db.restaurants.find({ name: /mon/i }, { name: 1, borough: 1, "address.coord": 1, cuisine: 1 })
```

OUTPUT:



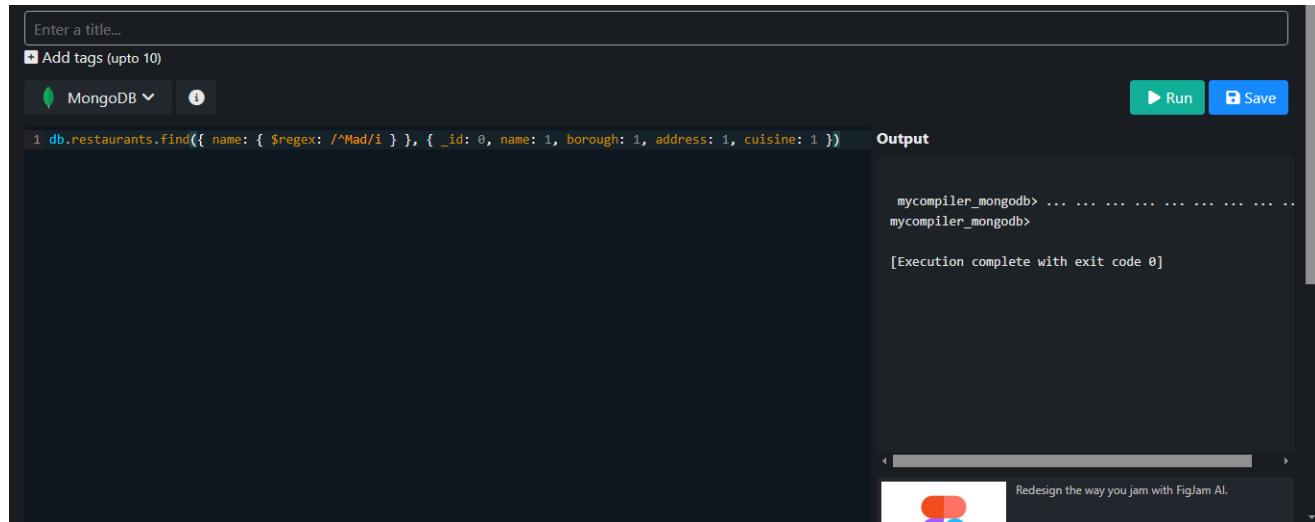
The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there are two dropdown menus: "MongoDB" and "Run". To the right of these are "Run" and "Save" buttons. The main area contains a command line with the following text:
1 db.restaurants.find({ name: { \$regex: /mon/i } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })
The output window on the right shows the command being run and the response:
mycompiler_mongodb> ...
mycompiler_mongodb>
[Execution complete with exit code 0]
At the bottom of the interface, there is a watermark for "Redesign the way you jam with FigJam AI".

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.

QUERY:

```
db.restaurants.find({ name: /^Mad/i }, { name: 1, borough: 1, "address.coord": 1, cuisine: 1 })
```

OUTPUT:



The screenshot shows a MongoDB shell interface. On the left, there is a code editor with the following query:

```
1 db.restaurants.find({ name: { $regex: /^Mad/i } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })
```

On the right, the 'Output' panel shows the results of the query execution:

```
mycompiler_mongodb> ... . . . . . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

At the bottom right of the output panel, there is a small advertisement for FigJam AI.

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

QUERY:

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

OUTPUT:



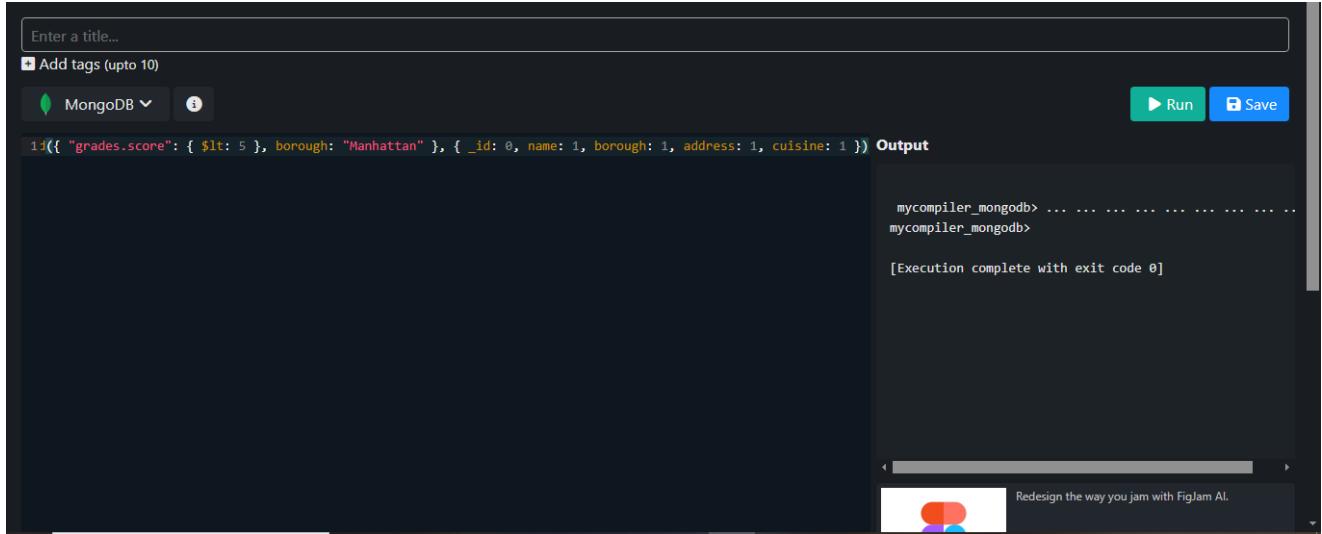
The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and a help icon. On the right side, there are "Run" and "Save" buttons. The main area contains a command line with the following text:
1 db.restaurants.find({ "grades.score": { \$lt: 5 } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })
To the right of the command line is a "Output" panel. The output shows the command being run and the response from the database:
mycompiler_mongodb>
mycompiler_mongodb>
[Execution complete with exit code 0]
At the bottom right of the interface, there is a watermark for FigJam AI.

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

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB query editor interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and an "i" button. To the right are "Run" and "Save" buttons. The main area contains a code editor with the following query:

```
1:({ "grades.score": { $lt: 5 }, borough: "Manhattan" }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })
```

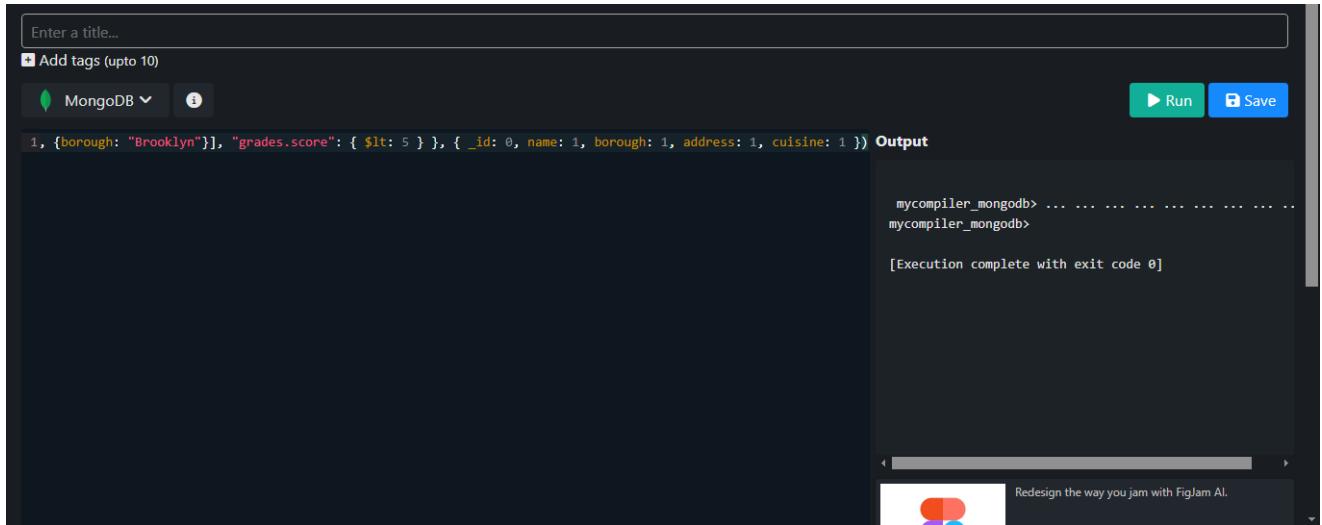
Below the code editor is a "Output" panel. It displays the command "mycompiler_mongodb> ..." followed by the result of the query: "mycompiler_mongodb> [Execution complete with exit code 0]". A progress bar is shown below the output panel, and at the bottom right, there is a watermark for "Redesign the way you jam with FigJam AI".

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

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there is a dropdown menu set to "MongoDB" and a help icon. On the right side, there are "Run" and "Save" buttons. The main area displays the query and its output. The query is:

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

The output shows the results of the query:

```
1, {borough: "Brooklyn"}, {"grades.score": { $lt: 5 } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 }
```

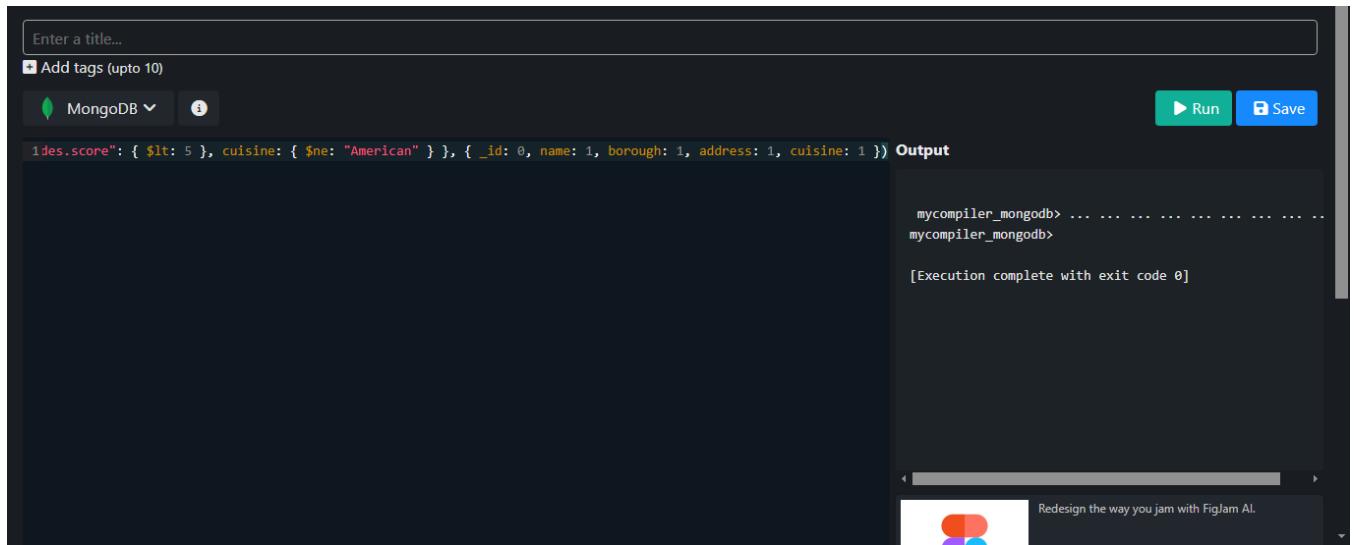
Below the output, the MongoDB prompt "mycompiler_mongodb>" is visible, followed by several ellipses indicating continuation. The message "[Execution complete with exit code 0]" is displayed at the bottom of the output window. A watermark for "Redesign the way you jam with FigJam AI." is visible at the bottom right of the interface.

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

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar are two dropdown menus: "MongoDB" and "Run". To the right of the dropdowns are "Run" and "Save" buttons. The main area is titled "Output" and contains the following text:

```
1{des.score": { $lt: 5 }, cuisine: { $ne: "American" } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 } } Output
```

```
mycompiler_mongodb> .....  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

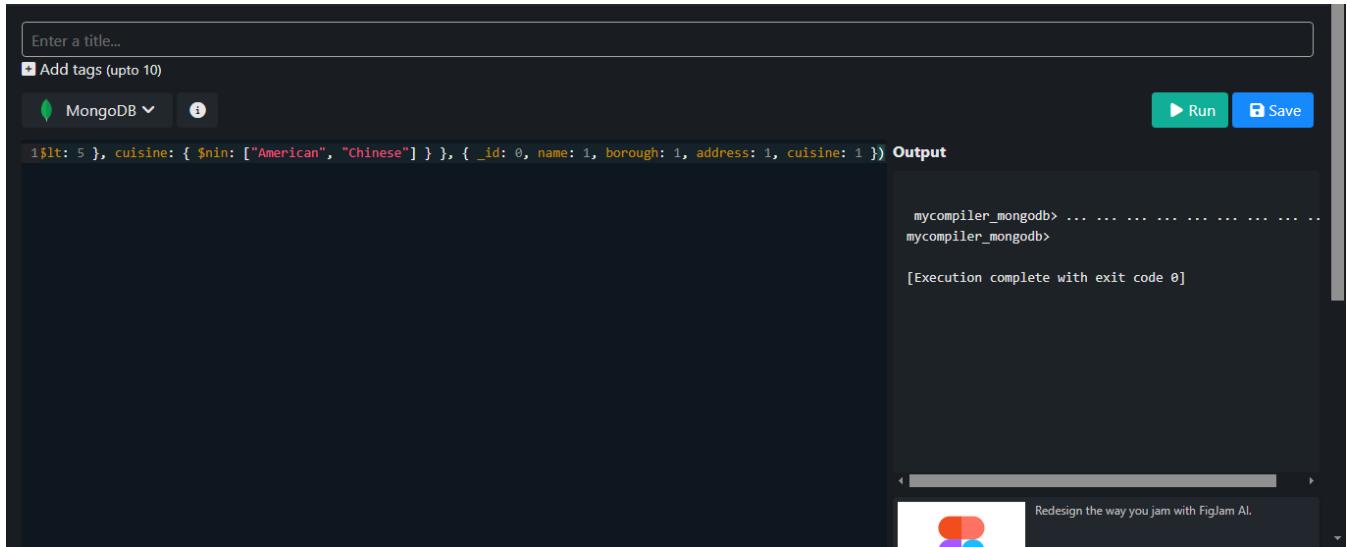
At the bottom right of the interface, there is a small advertisement for FigJam AI.

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

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB query editor interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there are two dropdown menus: "MongoDB" and "Run". To the right of the dropdowns are "Run" and "Save" buttons. The main area contains the MongoDB query:`db.restaurants.find({ "grades": { $elemMatch: { "score": { $lt: 5 } } }, $or: [{ "borough": "Manhattan" }, { "borough": "Brooklyn" }], "cuisine": { $nin: ["American", "Chinese"] } })`

Below the query, the word "Output" is followed by the command "mycompiler_mongodb>". The output window shows the results of the query execution:`mycompiler_mongodb>
mycompiler_mongodb>
[Execution complete with exit code 0]`

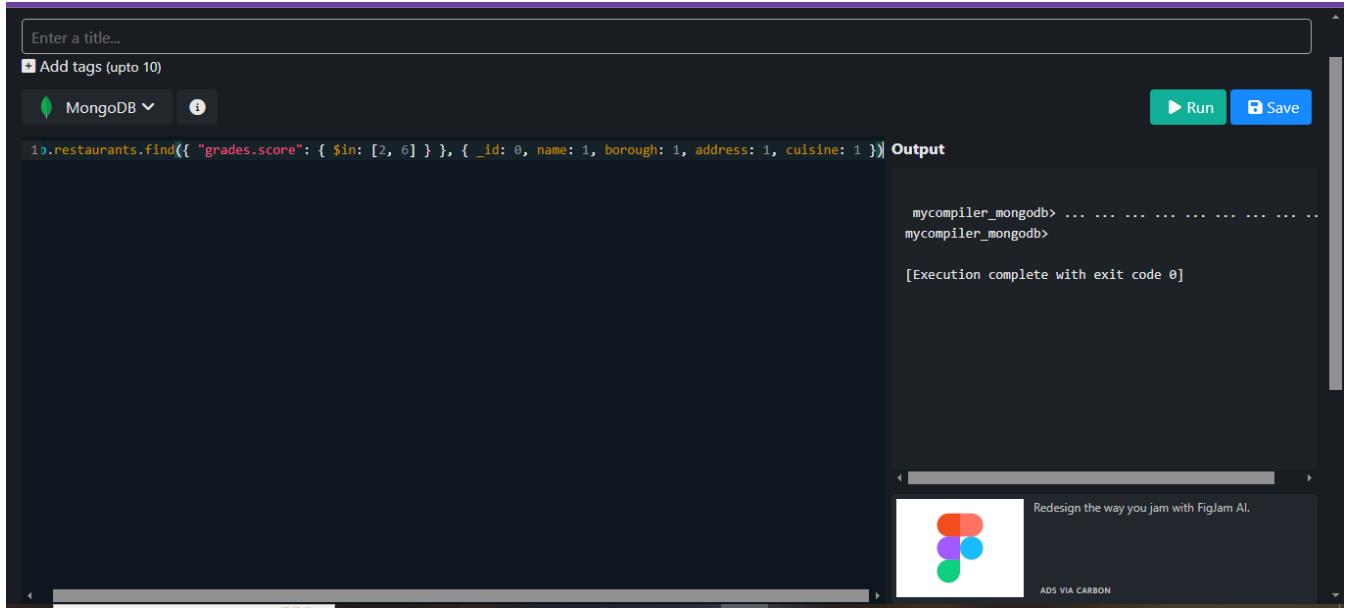
At the bottom of the interface, there is a watermark for FigJam AI.

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

QUERY:

```
db.restaurants.find({ $and: [{ "grades.grade": "A", "grades.score": 2 }, { "grades.grade": "A", "grades.score": 6 }] })
```

OUTPUT:



The screenshot shows a MongoDB shell interface. On the left, there is a search bar labeled "Enter a title..." and a "MongoDB" dropdown menu. Below the search bar, there is a "Run" button and a "Save" button. In the main area, a command is entered: `db.restaurants.find({ "grades.score": { $in: [2, 6] } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })`. To the right of the command, the word "Output" is displayed. The output window shows the results of the query execution. It starts with the prompt "mycompiler_mongodb>". The results are listed as follows:

```
mycompiler_mongodb> db.restaurants.find({ "grades.score": { $in: [2, 6] } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })
[Execution complete with exit code 0]
```

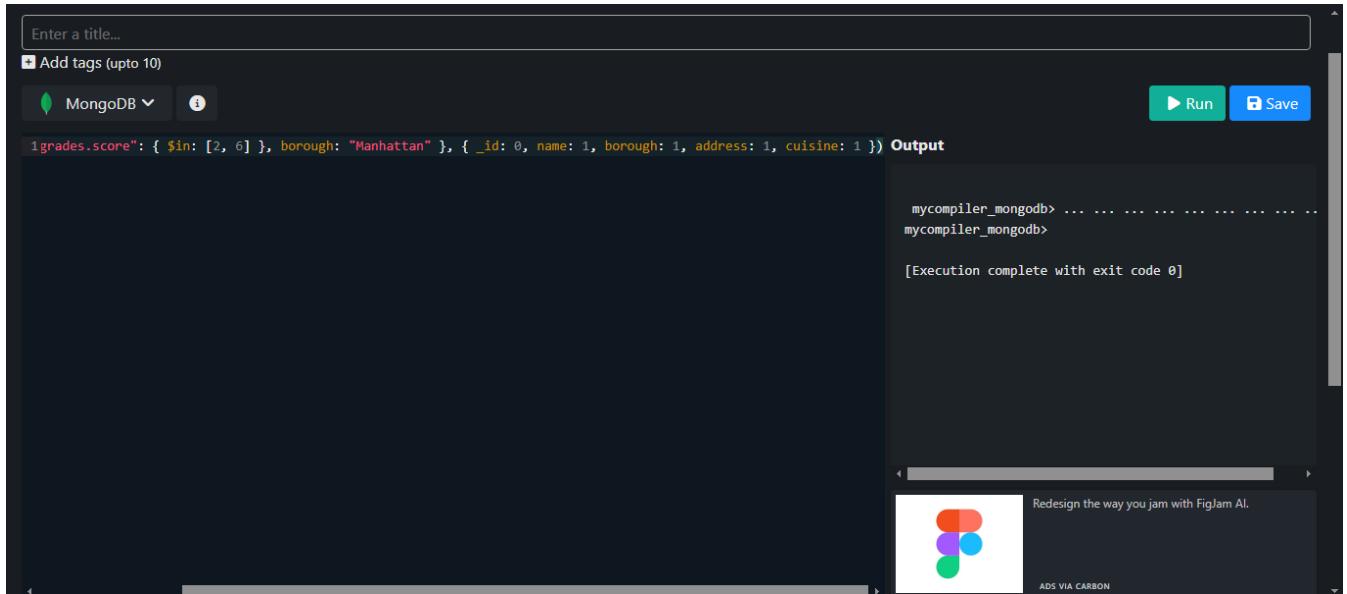
At the bottom right of the interface, there is an advertisement for FigJam AI.

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.

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a "Add tags (upto 10)" button. Below the search bar, there are two dropdown menus: "MongoDB" and "Save". On the right side, there are "Run" and "Save" buttons. The main area contains a command line and its output. The command is:

```
1grades.score": { $in: [2, 6] }, borough: "Manhattan" }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 }) Output
```

The output shows the results of the query execution:

```
mycompiler_mongodb> ... . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

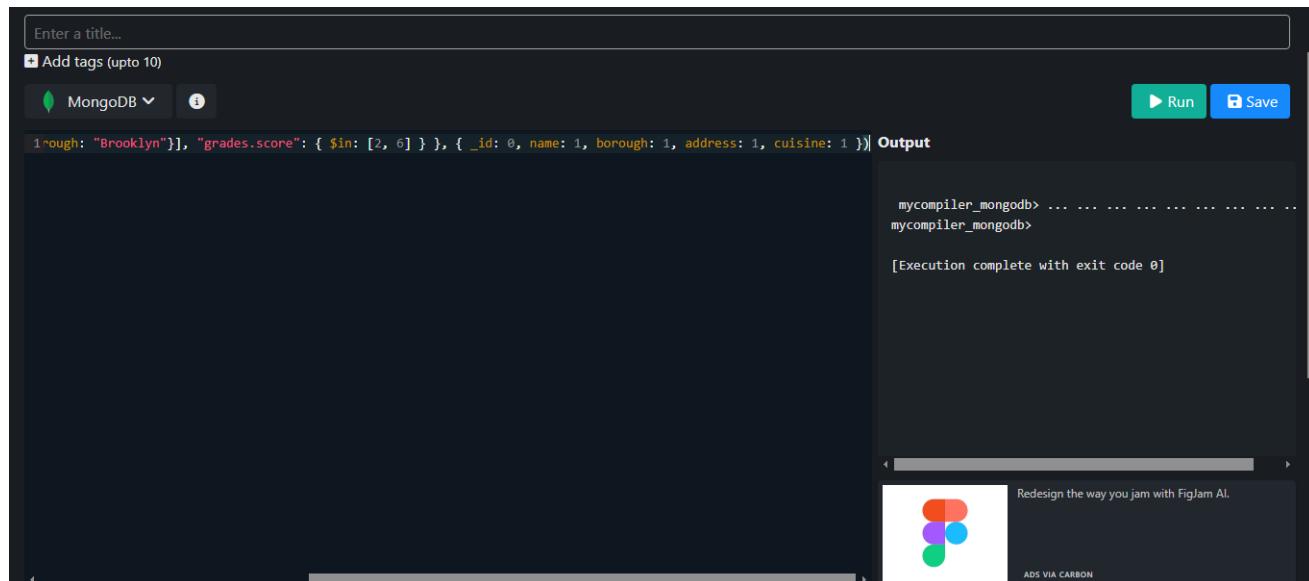
At the bottom right of the interface, there is an advertisement for FigJam AI.

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.

QUERY:

```
db.restaurants.find({ $and: [{ "grades.grade": "A", "grades.score": 2 }, { "grades.grade": "A", "grades.score": 6 }], $or: [{ "borough": "Manhattan" }, { "borough": "Brooklyn" }] })
```

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, the connection dropdown is set to "MongoDB" and there is an "Info" button. On the right side, there are "Run" and "Save" buttons. The main area contains a command line and its output. The command is:

```
db.restaurants.find({ $and: [{ "grades.grade": "A", "grades.score": 2 }, { "grades.grade": "A", "grades.score": 6 }], $or: [{ "borough": "Manhattan" }, { "borough": "Brooklyn" }] })
```

The output shows the results of the query:

```
mycompiler_mongodb> ... ... ... ... ...
mycompiler_mongodb>
[Execution complete with exit code 0]
```

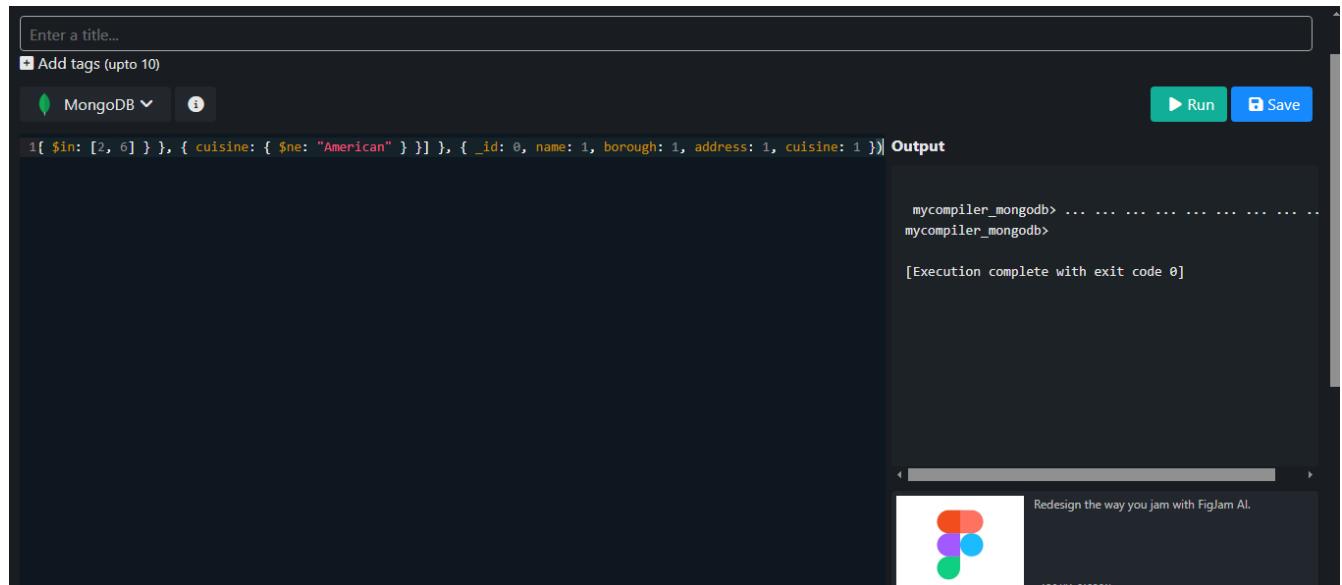
At the bottom of the interface, there is an advertisement for FigJam AI.

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.

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button for "Add tags (upto 10)". Below the search bar, there are two dropdown menus: one for "MongoDB" and another with an "i" icon. On the right side of the interface are two buttons: "Run" and "Save". The main area is titled "Output" and contains the following text:

```
{ $in: [2, 6] }, { cuisine: { $ne: "American" } } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })| Output
```

mycompiler_mongodb>
mycompiler_mongodb>
[Execution complete with exit code 0]

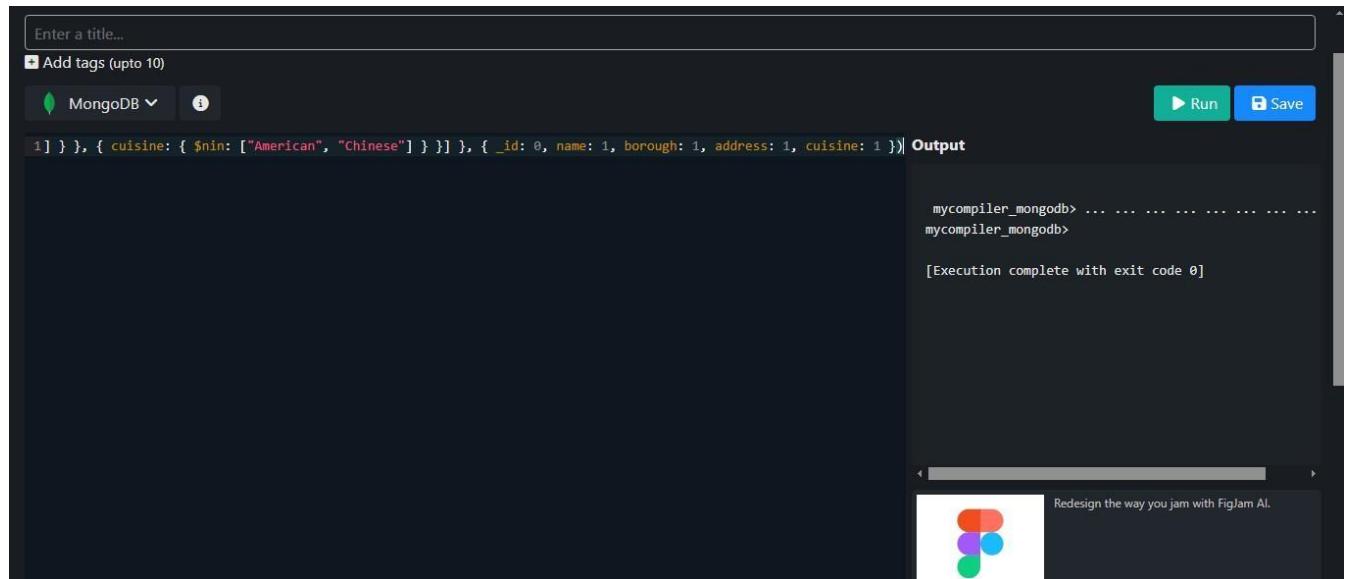
At the bottom right of the interface, there is an advertisement for FigJam AI with the text "Redesign the way you jam with FigJam AI." and "ADS VIA CARBON".

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.

QUERY:

```
db.restaurants.find({ $and: [{ "grades.grade": "A", "grades.score": 2 }, { "grades.grade": "A", "grades.score": 6 }], $or: [{ "borough": "Manhattan" }, { "borough": "Brooklyn" }], "cuisine": { $nin: ["American", "Chinese"] } })
```

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and a help icon. On the right side, there are "Run" and "Save" buttons. The main area contains the MongoDB command and its output. The command is:

```
1] } }, { cuisine: { $nin: ["American", "Chinese"] } } ] }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 }}) Output
```

The output shows the results of the query execution:

```
mycompiler_mongodb> ... ... ... ...
mycompiler_mongodb>
[Execution complete with exit code 0]
```

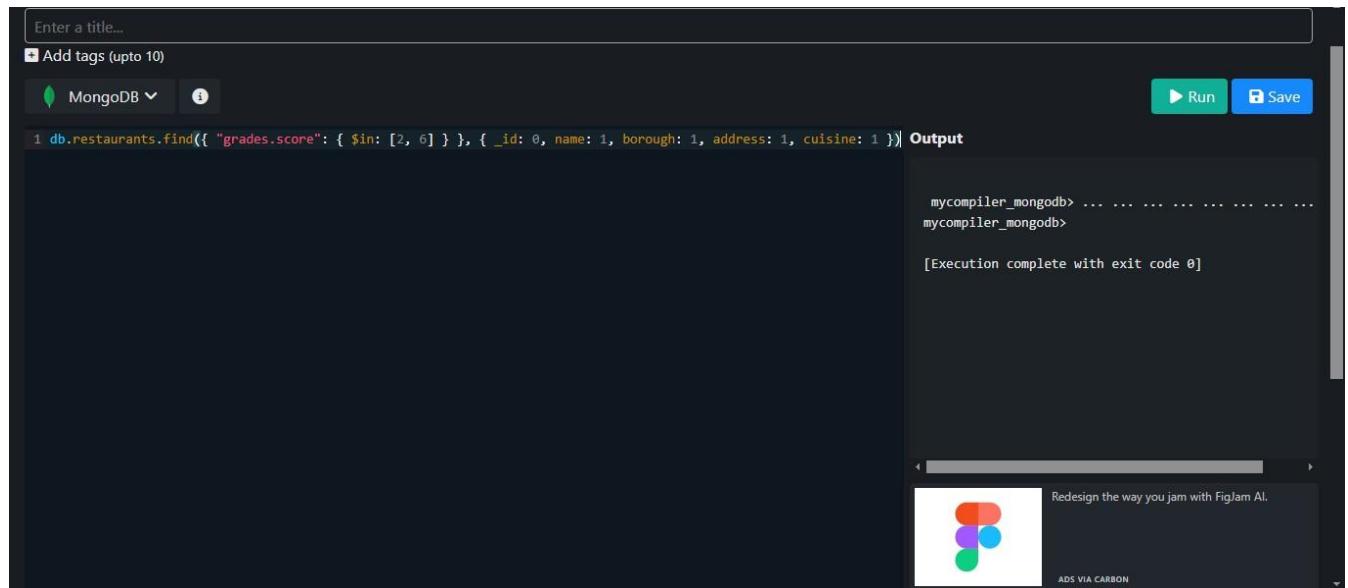
In the bottom right corner of the interface, there is a watermark for FigJam AI with the text "Redesign the way you jam with FigJam AI."

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.

QUERY:

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

OUTPUT:



The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there are two tabs: "MongoDB" (selected) and "Carbon". On the right side, there are "Run" and "Save" buttons. The main area displays a command and its output. The command is:

```
1 db.restaurants.find({ "grades.score": { $in: [2, 6] } }, { _id: 0, name: 1, borough: 1, address: 1, cuisine: 1 })| Output
```

The output shows the results of the query:

```
mycompiler_mongodb> ... ... ... ... ...
mycompiler_mongodb>
[Execution complete with exit code 0]
```

At the bottom right of the interface, there is an advertisement for FigJam AI.

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

MONGO DB

EX.NO:20

DATE: 17 - 5 - 24

REG.NO: 220701239

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

QUERY:

```
db.movies.find({ year: 1893 })
```

OUTPUT:

The screenshot shows a MongoDB query editor interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and an "i" button. On the right side of the top bar are "Run" and "Save" buttons. The main area contains a code input field with the following text:
`{age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 }`
To the right of the code input field is a "Output" tab. The output pane displays the command `mycompiler_mongodb>` followed by several dots, indicating a continuation of the command history. Below the command history is the message "[Execution complete with exit code 0]".

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

QUERY:

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

OUTPUT:

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

```
db.movies.find({ genres: 'Short' })
```

OUTPUT:

Enter a title...

Add tags (upto 10)

MongoDB ▾

Run Save

```
age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 })| Output
```

```
mycompiler_mongodb> ... . . . . .  
mycompiler_mongodb>  
[Execution complete with exit code 0]
```

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

QUERY:

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

OUTPUT:

The screenshot shows a MongoDB query interface. On the left, there's a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below these are two dropdown menus: "MongoDB" and "Output". To the right of the dropdowns are "Run" and "Save" buttons. The main area displays the query command: "db.movies.find({ directors: 'William K.L. Dickson' })". To the right of the command is the word "Output". The output window shows the results of the query, which consists of a single line of dots: "...". Below the output window, a message reads "[Execution complete with exit code 0]".

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

QUERY:

```
db.movies.find({ countries: 'USA' })
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and a help icon. On the right side, there are "Run" and "Save" buttons. The main area contains a code input field with the query: `1age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 })`. To the right of the code, the word "Output" is displayed in bold. The output window shows the command prompt: "mycompiler_mongodb> ...". Below the prompt, the text "mycompiler_mongodb>" is repeated. At the bottom of the output window, the message "[Execution complete with exit code 0]" is visible.

6.) Retrieve all movies from the 'movies' collection that have complete information and

are rated as "UNRATED".

QUERY:

```
db.movies.find({ rated: 'UNRATED' })
```

OUTPUT:

The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there are two buttons: "MongoDB" with a dropdown arrow and a help icon. On the right side of the interface, there are two buttons: "Run" (green) and "Save" (blue). The main area is titled "Output" and contains the command: "db.movies.find({ rated: 'UNRATED' })". The output of the command is shown below, starting with "mycompiler_mongodb> ...". The output continues with several dots and ends with "[Execution complete with exit code 0]".

```
db.movies.find({ rated: 'UNRATED' })| Output
mycompiler_mongodb> ...
mycompiler_mongodb>
[Execution complete with exit code 0]
```

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

QUERY:

```
db.movies.find({ 'imdb.votes': { $gt: 1000 } })
```

OUTPUT:

The screenshot shows a MongoDB shell interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there are two buttons: "MongoDB" with a dropdown arrow and an "i" button. On the right side, there are "Run" and "Save" buttons. The main area contains a command line and its output. The command is: `1 age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 })`. The output shows the prompt "mycompiler_mongodb>" followed by "[Execution complete with exit code 0]".

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

QUERY:

```
db.movies.find({ 'imdb.rating': { $gt: 7 } })
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and a help icon. To the right are "Run" and "Save" buttons. The main area contains the query text "db.movies.find({ 'imdb.rating': { \$gt: 7 } })" followed by an "Output" button. The output window shows the command "mycompiler_mongodb> db.movies.find({ 'imdb.rating': { \$gt: 7 } })" and the response "[Execution complete with exit code 0]".

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

QUERY:

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

OUTPUT:

Enter a title...

Add tags (upto 10)

MongoDB

Run Save

1age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 })} Output

mycompiler_mongodb> ...

[Execution complete with exit code 0]

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

QUERY:

```
db.movies.find({ 'awards.wins': { $gt: 0 } })
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there is a dropdown menu set to "MongoDB" and a help icon. On the right side, there are two buttons: "Run" (green) and "Save" (blue). The main area displays a query: "age: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 }" followed by the word "Output". To the right of the output field, the mongo shell prompt "mycompiler_mongodb>" is visible, along with several ellipsis dots (...). Below the prompt, the message "[Execution complete with exit code 0]" is displayed.

11.)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.

QUERY:

```
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 } )
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a checkbox for "Add tags (upto 10)". Below the search bar are two buttons: "MongoDB" with a dropdown arrow and a help icon. To the right are "Run" and "Save" buttons. The main area contains a query: "iges: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 })" followed by the word "Output". The output window shows the command "mycompiler_mongodb> ..." followed by "[Execution complete with exit code 0]".

12.)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".

QUERY:

```
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 } )
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below the search bar, there is a dropdown menu set to "MongoDB" and a help icon. On the right side, there are "Run" and "Save" buttons. The main area contains a code input field with the following query: `languages: 1, released: 1, directors: 1, writers: 1, awards: 1, year: 1, genres: 1, runtime: 1, cast: 1, countries: 1 }}, Output`. To the right of the code input, the word "Output" is bolded. The output pane below shows the command `mycompiler_mongodb> ...` followed by several dots, indicating a long list of results. At the bottom of the output pane, it says "[Execution complete with exit code 0]".

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

QUERY:

```
db.movies.find( { released: ISODate("1893-05-09T00:00:00.000Z") }, { title: 1, languages: 1, released: 1, directors: 1, writers: 1, countries: 1 } )
```

OUTPUT:

The screenshot shows a MongoDB query interface. At the top, there is a search bar labeled "Enter a title..." and a button "Add tags (upto 10)". Below this, there are two tabs: "MongoDB" (selected) and "Info". On the right side, there are "Run" and "Save" buttons. The main area displays a command and its output. The command is: `1sed.date": "May 9, 1893" }, { _id: 0, title: 1, languages: 1, released: 1, directors: 1, writers: 1, countries: 1 })`. The output shows the results of the query in the MongoDB shell, with the message "[Execution complete with exit code 0]" at the bottom.

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.

QUERY:

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

OUTPUT:

Enter a title...

Add tags (upto 10)

MongoDB  

`1s.find({ title: /scene/i }, { _id: 0, title: 1, languages: 1, released: 1, directors: 1, writers: 1, countries: 1 })`  

Output

```
mycompiler_mongodb> ... . . . . .
mycompiler_mongodb>

[Execution complete with exit code 0]
```

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