**PRACTICAL - 1**

**Aim:** Create Different Tables

**Theoretical Description:**

1) Create a schema for the bank that wants to keep the records of different Job

profile along with associated employee and their related information. Bank also

wants to keep the records of their account and loan related customer separately

in their database.

2) Create a schema for a new bank that wants to copy the entire structure from an

existing bank schema without data. It also wants to delete some of the copied

structure and alter the structure based on new nomenclature.

**Query-1:** Create Table Job (job\_id, job\_title, min\_sal, max\_sal)

**SQL Statement:**

Create Table Job(job\_id Varchar2(15), job\_title Varchar2(30), min\_sal Number(7,2), max\_sal Number(7,2));

select \* from job;

**Output:** Job table created.

**Query-2:** Create table Employee**.**

**SQL Statement:** Create Table Employee(emp\_no Number,emp\_name Varchar2(30),emp\_sal Number(\*,2),emp\_comm Number(6,1),dept\_no Number(3));

**Output:** Employee table created.

**Query-3:** Create table deposit.

**SQL Statement:** Create table deposit(a\_no Varchar2(20),cname Varchar2(30),bname Varchar2(10),amount Number(7,2),a\_date DATE);

**Output:** deposit table created.

**Query-4:** Create table borrow.

**SQL Statement:** Create table borrow(loanno Varchar2(5),cname Varchar2(30),bname Varchar2(10),amount Number(\*,2));

**Output:** borrow table created.

**Query-5:** Describe table Job,employee,deposit.

**SQL Statement:**

desc Job;

desc Employee;

desc deposit;

desc borrow;

**Output:** value inserted.

**Query-6:** Insert following values in the table Employee.

**SQL Statement:** INSERT INTO Employee(emp\_no, emp\_name, emp\_sal, emp\_comm, dept\_no) VALUES (101, 'Smith', 800, 5, 20),

INSERT INTO Employee VALUES(102, 'Snehal', 1600, 300, 25);

INSERT INTO Employee VALUES(103, 'Adama', 1100, 0, 20);

INSERT INTO Employee VALUES(104, 'Aman', 3000, 7, 15);

INSERT INTO Employee VALUES(105, 'Anita', 5000, 50000, 10);

INSERT INTO Employee VALUES(106, 'Sneha', 2450, 24500, 10);

INSERT INTO Employee VALUES(107, 'Anamika', 2975, 77, 30);

select \* from employee;

**Output:** value inserted.

**Query-7**: Insert following values in the table job.

**SQL Statement:**

INSERT ALL

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('IT\_PROG', 'Programmer', 4000, 10000)

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('MK\_MGR', 'Marketing manager', 9000, 15000)

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('FI\_MGR', 'Finance manager', 8200, 12000)

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('FI\_ACC', 'Account', 4200, 9000)

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('LEC', 'Lecturer', 6000, 17000)

INTO Job (job\_id, job\_title , min\_sal , max\_sal) values('COMP\_OP', 'Computer Operator', 1500, 3000)

SELECT \* FROM dual

**Output:** value inserted.

**Query-8:** Insert following values in the table deposit.

**SQL Statement:**

INSERT INTO deposit

values(&A\_no,'&cname','&Bname',&Amount,'&date');

select \* from deposit;

**Output:** value inserted.

**PRACTICAL - 2**

**Aim:** Perform Data Definition Language (DDL) commands and change the

existing schema as per given information.

**Theoretical Description**:

1) Create table supplier from employee with all the columns.

2) Create table sup1 from employee with first two columns.

3) Create table sup2 from employee with no data.

4) Insert the data into sup2 from employee whose name is ‘Anita’.

5) Rename the table sup2.

6) Destroy table sup1 with all the data.

7) Add one column phone to employee with size of column is Varchar2(10).

8) Modify column phone and change type to char(10).

9) Delete employee\_name column from sup2;

10) Rename the column salary to new\_sal in sup2;

**Query-1:** Create table supplier from employee with all the columns.

**SQL Statement:**

CREATE TABLE supplier as SELECT\*FROM employee;

SELECT \* FROM supplier;

**Output:** Table supplier created.

**Query-2:** Create table sup1 from employee with first two columns.

**SQL Statement:**

CREATE TABLE sup1 as SELECT emp\_no, emp\_name FROM employee;

SELECT \* FROM sup1;

**Output:** sup2 table is created.

**Query-3:** Create table sup2 from employee with no data.

**SQL Statement:**

CREATE TABLE sup2 AS SELECT \* FROM employee WHERE 1=0;

SELECT \* FROM sup2;

**Output:** sup2 table is created.

**Query-4:** Insert the data into sup2 from employee whose name is ‘Anita’.

**SQL Statement:**

INSERT INTO sup2 SELECT \* FROM employee WHERE emp\_name = 'Anita';

**Output:** Data is inserted.

**Query-5:** Rename the table sup2.

**SQL Statement:**

ALTER TABLE sup2 RENAME TO new\_sup2;

**Output:** table is rename to new\_sup2.

**Query-6:** Destroy table sup1 with all the data**.**

**SQL Statement:**

DROP TABLE sup1;

**Output:** table sup1 is destroyed with all data.

**Query-7:** Add one column phone to employee with size of column is Varchar2(10).

**SQL Statement:**

ALTER TABLE employee ADD phone VARCHAR2(10);

**Output:** one column added.

**Query-8:** Modify column phone and change type to char(10).

**SQL Statement:**

ALTER TABLE employee MODIFY phone CHAR(10);

**Output:** one column modified.

**Query-9:** Delete employee\_name column from sup2.

**SQL Statement:**

ALTER TABLE new\_sup2 DROP COLUMN emp\_name;

**Output:** column deleted.

**Query-10:** Rename the column salary to new\_sal in sup2;

**SQL Statement:**

ALTER TABLE new\_sup2 RENAME COLUMN emp\_sal TO new\_sal;

**Output:** column is rename to new\_sal.

**PRACTICAL - 3**

**Aim: To Perform Data Query Language (DQL) and Data Manipulation Language (DML) Commands.**

**Theoretical Description:**

1. **DQL (Data Query Language) :**

**DQL is used to retrieve data from a database without modifying it. The main command in DQL is SELECT.**

1. **SELECT: Retrieves data from a table.**
2. **DML (Data Manipulation Language) :**

**DML is used to manipulate data in the database. It includes commands like INSERT, UPDATE, and DELETE.**

1. **INSERT: Adds new data to a table.**
2. **UPDATE: Modifies existing data in a table.**
3. **DELETE: Removes data from a table.**

**Query-1: Retrieve all data from employee, jobs and deposit.**

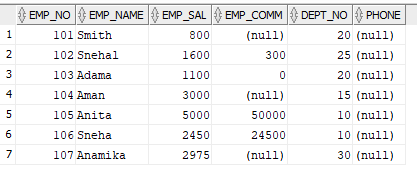
**SQL Statement:**

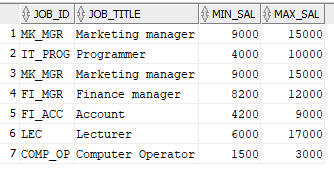
**select \* from Employee;**

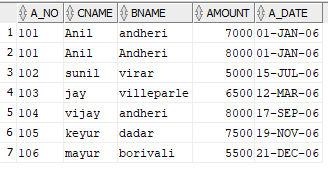
**select \* from Job;**

**select \* from deposit;**

**Output:**

****

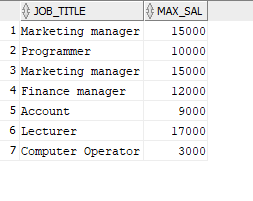
****

****

**Query-2: Display job title and maximum salary of all jobs.**

**SQL Statement: select job\_title,max\_sal from Job ;**

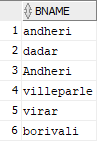
**Output:**

****

**Query-3: Write a query to find out to all the branches.**

**SQL Statement:**

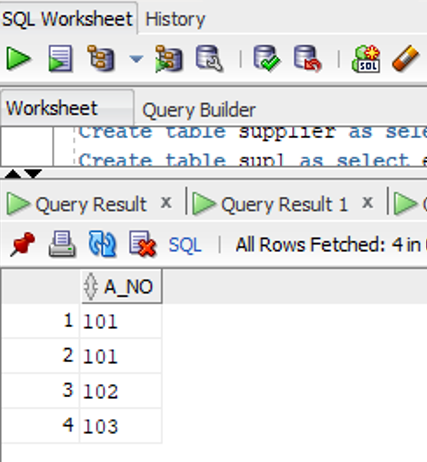
**Output:**

****

**Query-4: Display all the account no. into which rupees are between dates 01-01-06 and 25-07-06.**

**SQL Statement: select a\_no from deposit where a\_date between '01-JAN-2006' AND '25-JULY-2006';**

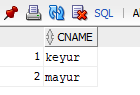
**Output:**

****

**Query-5: Display names of all customers whose account is deposited after 09-oct-06**

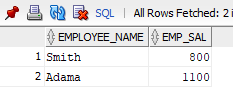
**SQL Statement: SELECT cname FROM deposit WHERE a\_date > '09-OCT-2006';**

**Output:**

****

**Query-6: Display name and salary of employee whose department no is 20. Give alias name to name of employee.**

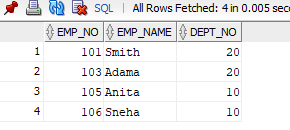
**SQL Statement: SELECT emp\_name AS employee\_name, emp\_sal FROM Employee WHERE dept\_no = 20;**

**Output:   
**

**Query-7: Display employee no, name and department details of those employee whose department lies in(10,20).**

**SQL Statement: SELECT emp\_no, emp\_name, dept\_no FROM Employee WHERE dept\_no IN (10, 20);**

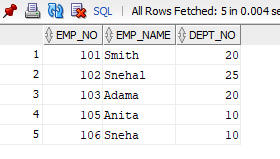
**Output:**

****

**Query-8: Display employee no, name and department details of those employee whose department not in(15,30).**

**SQL Statement: SELECT emp\_no, emp\_name, dept\_no FROM employee WHERE dept\_no NOT IN (15, 30);**

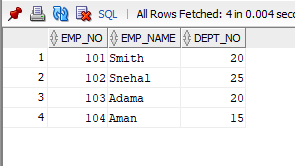
**Output:**

****

**Query-9: Display employee no, name and department details of those employee whose department no is between 15 and 25.**

**SQL Statement: SELECT emp\_no, emp\_name, dept\_no FROM employee WHERE dept\_no BETWEEN 15 AND 25;**

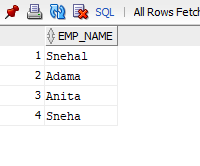
**Output:**

****

**Query-10: Display name of all employee whose emp\_comm contains the non-null values.**

**SQL Statement: SELECT emp\_name FROM employee WHERE emp\_comm IS NOT NULL;**

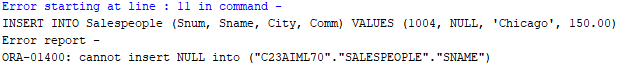
**Output:**

****

**Query-11: Combine two columns min\_sal and max\_sal and display it one column using common alias name.**

**SQL Statement:**

**Output:**



**Query-12: Insert the data into sup2 from employee.**

**SQL Statement: INSERT INTO sup2 SELECT \* FROM Employee;**

**Output: 2 rows inserted**

**Query-13: Delete all the rows from sup1 as sup.**

**SQL Statement: DELETE FROM sup1;**

**Output: 7 rows deleted.**

**Query-14: Delete the detail of supplier whose emp\_no is 103.**

**SQL Statement: DELETE FROM supplier WHERE emp\_no = 103;**

**Output: 1 row deleted.**

**Query-15: Update the name of employee to ‘Aman’ name whose emp\_name is ‘Anita’.**

**SQL Statement: UPDATE employee SET emp\_name = 'Aman' WHERE emp\_name = 'Anita';**

**Output: 1 row updated.**

**Query-16: Update the value of employee name whose employee number is 103.**

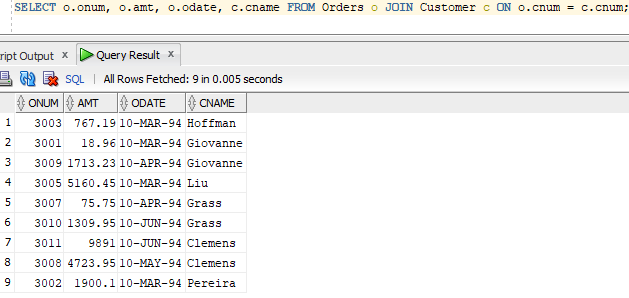
**SQL Statement: UPDATE employee SET emp\_name = 'New Name' WHERE emp\_no = 103;**

**Output: 1 row updated.**

**Query-17: Find out the maximum and minimum salary form job table.**

**SQL Statement: SELECT MAX(max\_sal), MIN(min\_sal) FROM jobs;**

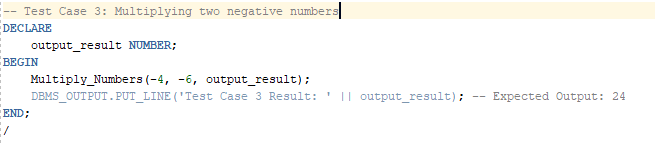
**Output:**

****

**Query-18: Find out the average salary of employee.**

**SQL Statement: SELECT AVG(emp\_sal) FROM employee;**

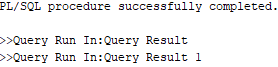
**Output:**

****

**Query-19: Count the total no as well as distinct rows in dept\_no column with a condition of salary greater than 1000 of employee.**

**SQL Statement: SELECT COUNT(DISTINCT dept\_no) FROM employee WHERE emp\_sal > 1000;**

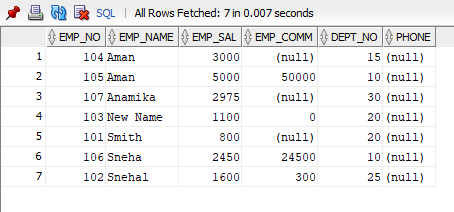
**Output:**

****

**Query-20: Display the detail of all employees in ascending order, descending order of their name and no.**

**SQL Statement: SELECT \* FROM employee ORDER BY emp\_name ASC, emp\_no ASC;**

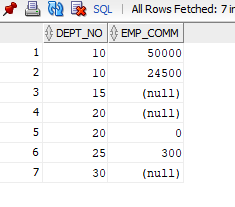
**Output:**

****

**Query-21:Display the dept\_no in ascending order and accordingly display emp\_comm in descending order.**

**SQL Statement: SELECT dept\_no, emp\_comm FROM employee ORDER BY dept\_no ASC, emp\_comm DESC;**

**Output:**

****

**Query-22: Update the value of emp\_comm to 500 where dept\_no is 20.**

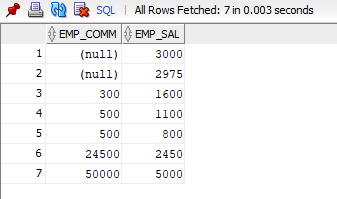
**SQL Statement: UPDATE employee SET emp\_comm = 500 WHERE dept\_no = 20;**

**Output: 2 rows updated.**

**Query-23: Display the emp\_comm in ascending order with null value first and accordingly sort employee salary in descending order.**

**SQL Statement: SELECT emp\_comm, emp\_sal FROM employee ORDER BY emp\_comm ASC NULLS FIRST, emp\_sal DESC;**

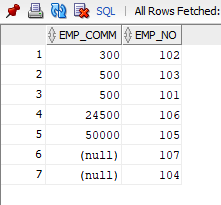
**Output:**

****

**Query-24: Display the emp\_comm in ascending order with null value last and accordingly sort emp\_no in descending order.**

**SQL Statement: SELECT emp\_comm, emp\_no FROM employee ORDER BY emp\_comm ASC NULLS LAST, emp\_no DESC;**

**Output:**

****

**// [IF any Questions are given to you]**

**Question-1:**

**Answer:**

**Question-2:**

**Answer:**

**PRACTICAL - 4**

**Aim: Execute the value matching and pattern matching conditions on the bank’s schema to satisfy the following requirements:**

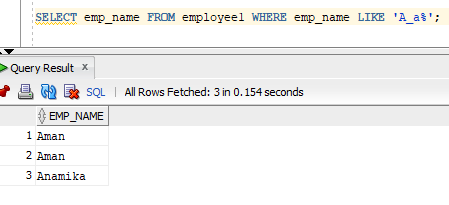
**Theoretical Description:**

1. Used to **retrieve data** from the database without modifying it. The primary command is **SELECT**, which allows querying data based on conditions.
2. The **LIKE** operator is used for **pattern matching** in string values. It allows us to search for data that matches a particular pattern, such as certain characters at specific positions in a string.
3. **Value Matching**: Used to find exact matches or ranges in values (e.g., checking branch names or salary values).
4. **Pattern Matching**: Used to find data based on patterns in strings (e.g., employee names, job IDs).
5. Each task uses the SELECT statement with **LIKE** for pattern matching and simple operators for value matching. This allows filtering employees, customers, and jobs based on the given conditions.
6. The ESCAPE keyword allows treating special characters (like \_ and %) as literal characters in pattern matching. For example, '\\\_%' will search for an underscore (\_) in the string.

**Query-1:Display all employee whose name start with ‘A’ and third character is ‘a’.**

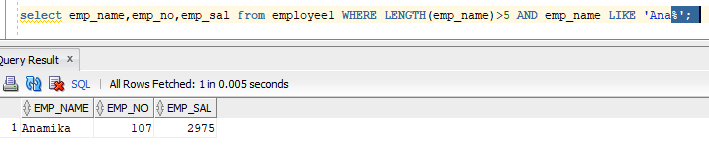
**SELECT emp\_name FROM employee1 WHERE emp\_name LIKE 'A\_a%';**

**Output:**

****

**Query-2: Display name, number and salary of those employees whose name is 5 characters long and first three characters are ‘Ana’.**

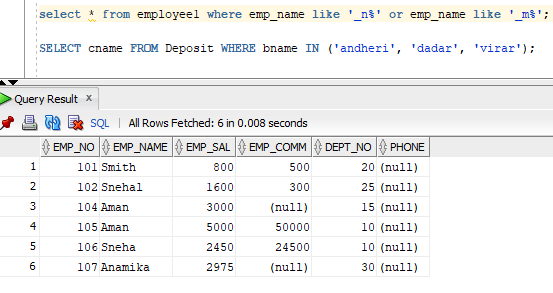
**select emp\_name,emp\_no,emp\_sal from employee1 WHERE LENGTH(emp\_name)>5 AND emp\_name LIKE 'Ana%';**

**Output: **

**Query-3: Display all information of employee whose second character of name is either ‘M’ or ‘N’.**

**SQL Statement: select \* from employee1 where emp\_name like '\_n%' or emp\_name like '\_m%';**

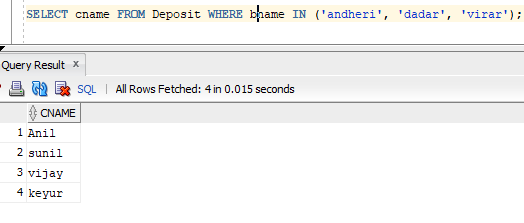
**Output:**

****

**Query-4: Find the list of all customer name whose branch is in ‘andheri’ or ‘dadar’ or ‘virar’.**

**SQL Statement: SELECT cname FROM Deposit WHERE bname IN ('andheri', 'dadar', 'virar');**

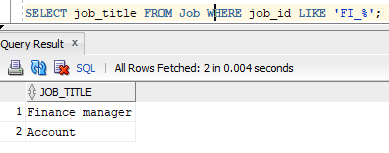
**Output:**

****

**Query-5: Display the job name whose first three character in job id field is ‘FI\_’.**

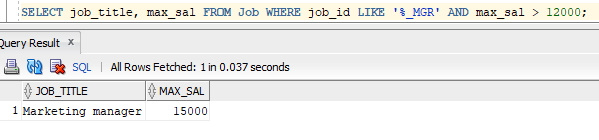
**SQL Statement: SELECT job\_title FROM Job WHERE job\_id LIKE 'FI\_%';**

**Output:**

****

**Query-6: Display the title/name of job whose last three character are ‘\_MGR’ and there maximum salary is greater than Rs 12000.**

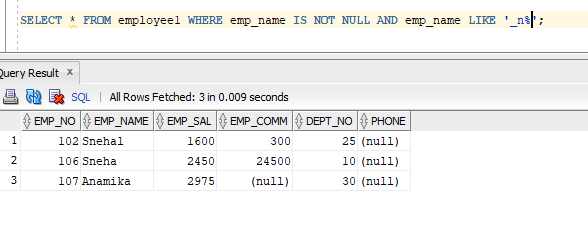
**SQL Statement: SELECT job\_title, max\_sal FROM Job WHERE job\_id LIKE '%\_MGR' AND max\_sal > 12000;**

**Output:   
**

**Query-7: Display the non-null values of employees and also employee name second character should be ‘n’ and string should be 5 character long.**

**SQL Statement: SELECT \* FROM employee1 WHERE emp\_name IS NOT NULL AND emp\_name LIKE '\_n%';**

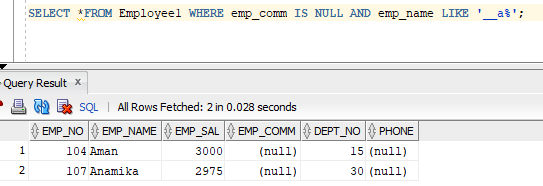
**Output:**

****

**Query-8: Display the null values of employee and also employee name’s third character should be ‘a’.**

**SQL Statement: SELECT \*FROM Employee1 WHERE emp\_comm IS NULL AND emp\_name LIKE '\_\_a%';**

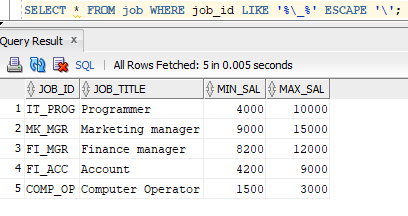
**Output:**

****

**Query-9: Display employee no, name and department details of those employee whose department no is between 15 and 25.**

**SQL Statement: SELECT \* FROM job WHERE job\_id LIKE '%\\_%' ESCAPE '/';**

**Output:**

****

**Question-1: how to show first five row of table?**

**Answer:** SQL Statement : SELECT \* FROM table\_name WHERE ROWNUM <= 5;

**Question-2: how to delete duplicate records?**

**Answer**: SQL Statement : DELETE FROM table\_name WHERE ROWID NOT IN (SELECT MIN(ROWID) FROM table\_name GROUP BY column\_name);

**Question-3:difference between delete , drop and transcate.**

**Answer:**

1. **DELETE** → Removes selected rows (WHERE can be used), logs changes, can be rolled back.
2. **DROP** → Removes the table completely (structure + data), **cannot** be rolled back.
3. **TRUNCATE** → Removes all rows quickly, keeps structure, **cannot** be rolled back in most databases.

**PRACTICAL -5**

**Aim:** To Study data communication using SQL functions

**Theoretical Description**:

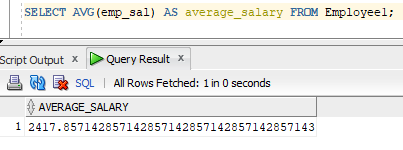
1. **AVG(n)** – Returns the average of numeric values.
2. **MIN(expr)** – Returns the minimum value from a column.
3. \**COUNT(expr / )* – Returns the number of rows.
4. **MAX(expr)** – Returns the maximum value from a column.
5. **SUM(n)** – Returns the total sum of numeric values.
6. **ABS(n)** – Returns the absolute value of n.
7. **POWER(m, n)** – Returns m raised to the power of n.
8. **ROUND(n, m)** – Rounds n to m decimal places.
9. **SQRT(n)** – Returns the square root of n.
10. **LOWER(char)** – Converts text to lowercase.
11. **INITCAP(char)** – Capitalizes the first letter of each word.
12. **UPPER(char)** – Converts text to uppercase.
13. **SUBSTR(char, m, n)** – Extracts a substring from position m of length n.
14. **LENGTH(word)** – Returns the length of a string.
15. **LTRIM(char, set)** – Removes leading characters from a string.
16. **RTRIM(char, set)** – Removes trailing characters from a string.
17. **LPAD(char1, n, char2)** – Pads char1 on the left with char2 to make it n characters long.
18. **RPAD(char1, n, char2)** – Pads char1 on the right with char2 to make it n characters long.
19. **TO\_NUMBER(char)** – Converts a string to a number.
20. **TO\_CHAR(n, fmt)** – Converts a number to a formatted string.
21. **TO\_DATE(char, fmt)** – Converts a string into a date format.
22. **ADD\_MONTHS(d, n)** – Adds n months to a given date d.
23. **LAST\_DAY(d)** – Returns the last day of the month for a given date.
24. **MONTHS\_BETWEEN(d1, d2)** – Finds the number of months between two dates.
25. **NEXT\_DAY(date, char)** – Returns the next occurrence of a specified weekday.
26. **UNION** – Combines unique results from two queries.
27. **UNION ALL** – Combines results from two queries (including duplicates).
28. **INTERSECT** – Returns common records from two queries.

**Aggregate Functions:**

**Query-1:** AVG (DISTINCT | ALL | n)

**SQL Statement:** SELECT AVG(emp\_sal) AS average\_salary FROM Employee1

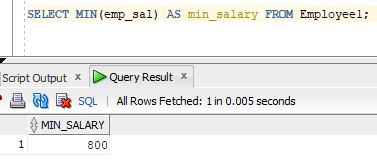
**Output:**

****

**Query-2:** MIN (DISTINCT | ALL | expr)

**SQL Statement:** SELECT MIN(emp\_sal) AS min\_salary FROM Employee1;

**Output:**

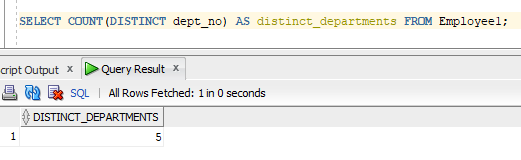
****

**Query-3:** COUNT (DISTINCT | ALL | expr)

**SQL Statement:**

SELECT COUNT(DISTINCT dept\_no) AS distinct\_departments FROM Employee1;

**Output:**

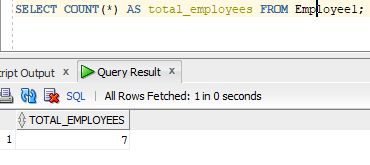


**Query-4:** COUNT (\*)

**SQL Statement:**

SELECT COUNT(\*) AS total\_employees FROM Employee1;

**Output:**

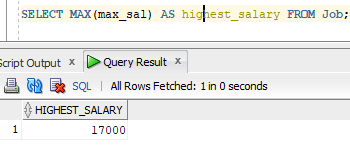
****

**Query-5:** MAX (DISTINCT | ALL | expr)

**SQL Statement:**

SELECT MAX(max\_sal) AS highest\_salary FROM Job;

**Output:**

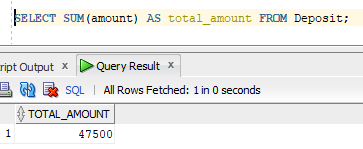
****

**Query-6:** SUM (DISTINCT | ALL | n)

**SQL Statement:**

SELECT SUM(amount) AS total\_amount FROM Deposit;

**Output:**

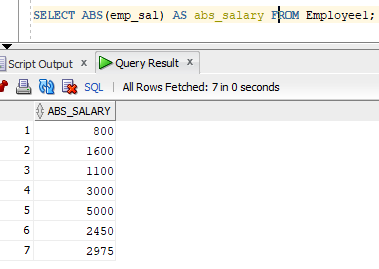
****

**Query-7:** Numeric Functions:ABS(n)

**SQL Statement:**

SELECT ABS(emp\_sal) AS abs\_salary FROM Employee1;

**Output:**

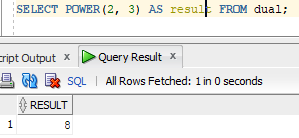


**Query-8:** POWER(m, n)

**SQL Statement:**

SELECT POWER(2, 3) AS result FROM dual;.

**Output:**

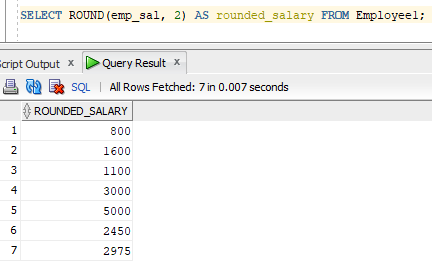


**Query-9:** ROUND(n, m)

**SQL Statement:**

SELECT ROUND(emp\_sal, 2) AS rounded\_salary FROM Employee1;

**Output:**

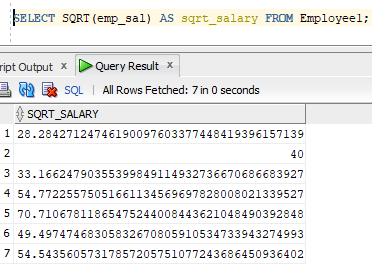
****

**Query-10:** SQRT(n)

**SQL Statement:**

SELECT SQRT(emp\_sal) AS sqrt\_salary FROM Employee1;

**Output:**

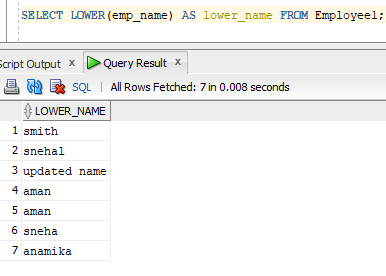
****

**Query-11:** String Functions: LOWER(char)

**SQL Statement:**

SELECT LOWER(emp\_name) AS lower\_name FROM Employee1;

**Output:**

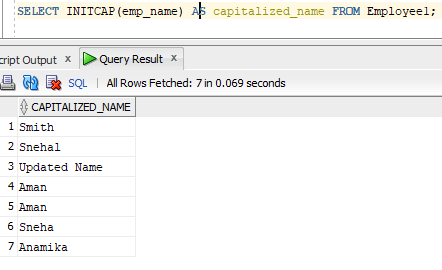
****

**Query-12:** INITCAP(char)

**SQL Statement:**

SELECT INITCAP(emp\_name) AS capitalized\_name FROM Employee1;

**Output:**

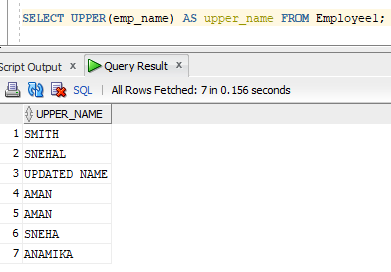


**Query-13:** UPPER(char)

**SQL Statement:**

SELECT UPPER(emp\_name) AS upper\_name FROM Employee1;

**Output:**

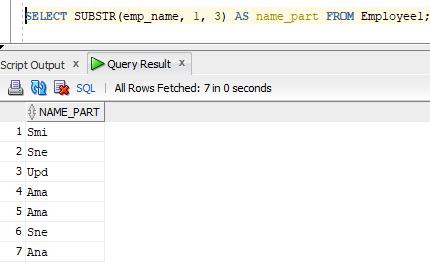
****

**Query-14:** SUBSTR(char, m [, n])

**SQL Statement:**

SELECT SUBSTR(emp\_name, 1, 3) AS name\_part FROM Employee1;

**Output:**

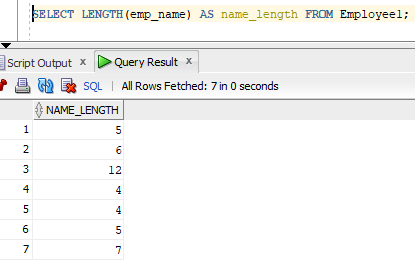
****

**Query-15:** LENGTH(word)

**SQL Statement:**

SELECT LENGTH(emp\_name) AS name\_length FROM Employee1;

**Output:**

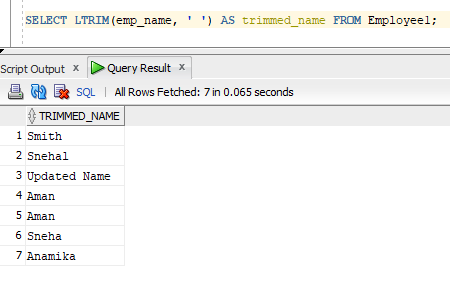
****

**Query-16:** LTRIM(char [, set])

**SQL Statement:**

SELECT LTRIM(emp\_name, ' ') AS trimmed\_name FROM Employee1;

**Output:**

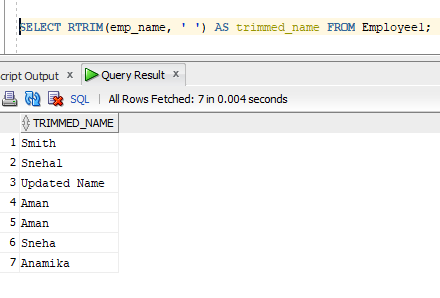
****

**Query-17:** RTRIM(char [, set])

**SQL Statement:**

SELECT RTRIM(emp\_name, ' ') AS trimmed\_name FROM Employee1;

**Output:**

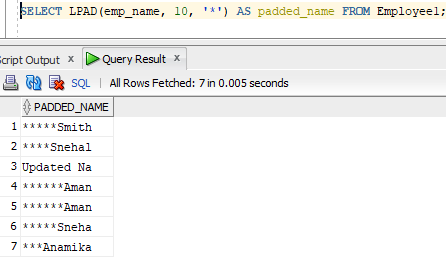
****

**Query-18:** LPAD(char1, n [, char2])

**SQL Statement:**

SELECT LPAD(emp\_name, 10, '\*') AS padded\_name FROM Employee1;

**Output:**

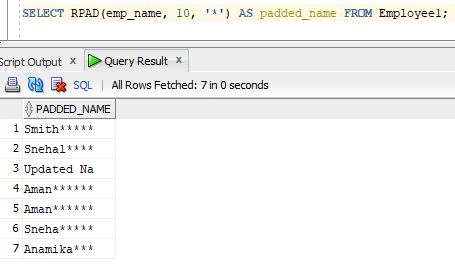
****

**Query-19:** RPAD(char1, n [, char2])

**SQL Statement:**

SELECT RPAD(emp\_name, 10, '\*') AS padded\_name FROM Employee1;

**Output:**

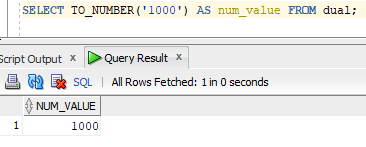
****

**Query-20: Conversion Functions:**TO\_NUMBER(char)

**SQL Statement:**

SELECT TO\_NUMBER('1000') AS num\_value FROM dual;

**Output:**

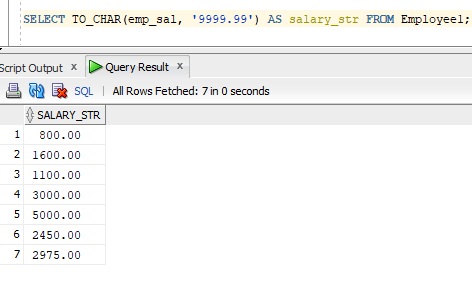
****

**Query-21:** TO\_CHAR(n [, fmt])

**SQL Statement:**

SELECT TO\_CHAR(emp\_sal, '9999.99') AS salary\_str FROM Employee1;

**Output:**

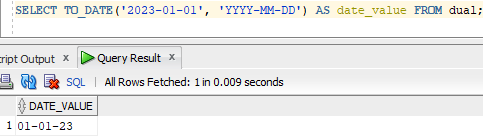
****

**Query-22:** Date Conversion Functions: TO\_DATE (char [, fmt])

**SQL Statement:**

SELECT TO\_DATE('2023-01-01', 'YYYY-MM-DD') AS date\_value FROM dual;

**Output:**

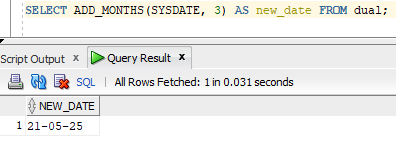
****

**Query-23:** DATE Functions: ADD\_MONTHS(d, n)

**SQL Statement:**

SELECT ADD\_MONTHS(SYSDATE, 3) AS new\_date FROM dual;

**Output:**

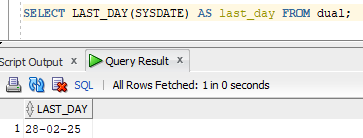
****

**Query-24:** LAST\_DAY(d)

**SQL Statement:**

SELECT LAST\_DAY(SYSDATE) AS last\_day FROM dual;

**Output:**

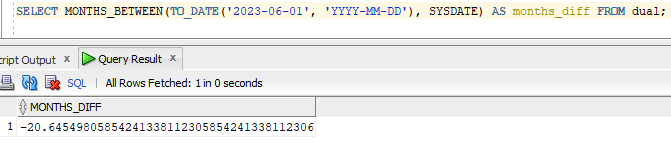
****

**Query-25:** MONTHS\_BETWEEN(d1, d2)

**SQL Statement:**

SELECT MONTHS\_BETWEEN(TO\_DATE('2023-06-01', 'YYYY-MM-DD'), SYSDATE) AS months\_diff FROM dual;

**Output:**

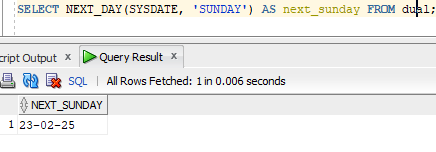
****

**Query-26:** NEXT\_DAY(date, char)

**SQL Statement:**

SELECT NEXT\_DAY(SYSDATE, 'SUNDAY') AS next\_sunday FROM dual;

**Output:**

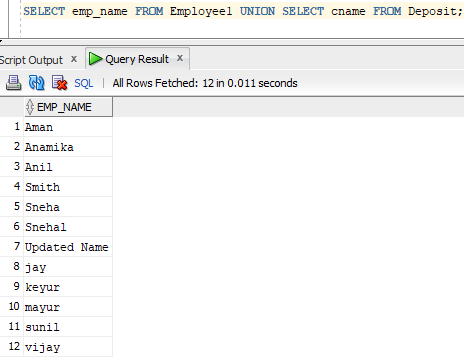
****

**Query-27:** UNION

**SQL Statement:**

SELECT emp\_name FROM Employee1 UNION SELECT cname FROM Deposit;

**Output:**

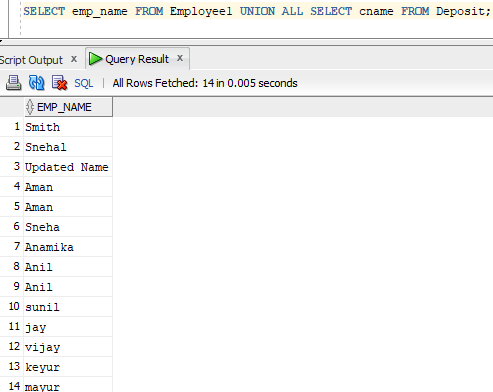
****

**Query-28:** UNION ALL

**SQL Statement:**

SELECT emp\_name FROM Employee1 UNION ALL SELECT cname FROM Deposit;

**Output:**

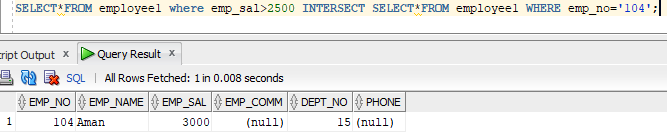
****

**Query-29:** INTERSECTION

**SQL Statement:**

SELECT FROM employeel where emp\_sal>2500 INTERSECT SELECT FROM employeel WHERE emp\_no='104';

**Output:**

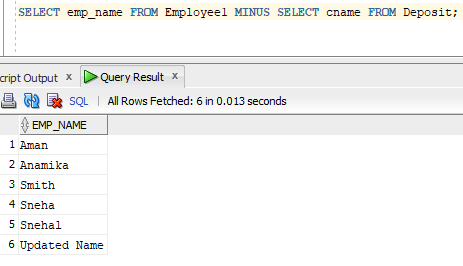
****

**Query-30:** MINUS

**SQL Statement:**

SELECT emp\_name FROM Employee1 MINUS SELECT cname FROM Deposit;

**Output:**

****

**PRACTICAL - 6**

**Aim: To solve various queries related to grouping and aggregate functions**

**Theoretical Description:**

1. **Aggregate Functions**: SUM(), AVG(), MAX(), etc., are used to perform calculations on groups of rows.
2. **GROUP BY**: Groups rows that have the same values in specified columns, typically used with aggregate functions.
3. **HAVING**: Filters groups based on the result of aggregate functions, applied after GROUP BY.
4. **WHERE**: Filters rows before grouping or aggregation.

**Query-1: 1)**

**Insert the following values into product table.**

**SQL Statement:**

CREATE TABLE product (Detorder\_no Varchar(6),Product\_no Varchar(6),Qty\_order INT);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19001', 'P00001', 10);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19001', 'P00002', 3);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19002', 'P00001', 4);

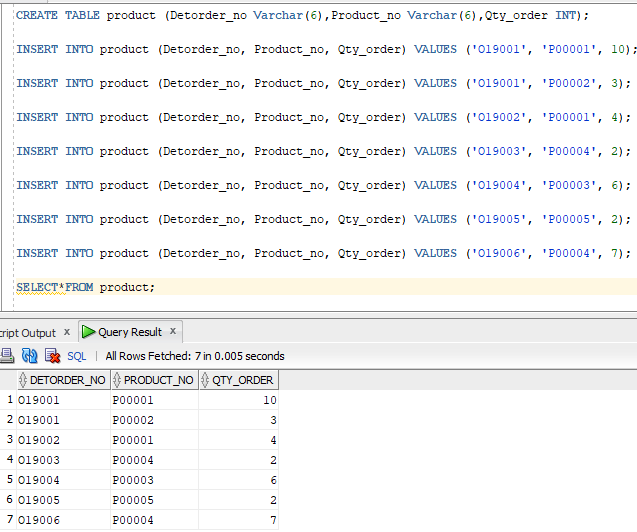
INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19003', 'P00004', 2);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19004', 'P00003', 6);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19005', 'P00005', 2);

INSERT INTO product (Detorder\_no, Product\_no, Qty\_order) VALUES ('O19006', 'P00004', 7);

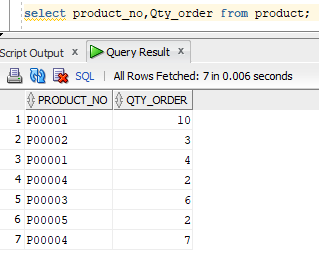
**Output:**

****

**Query-2: Retrieve the product numbers and total quantity ordered for each product from the product table.**

**SQL Statement:** select product\_no,Qty\_order from product;

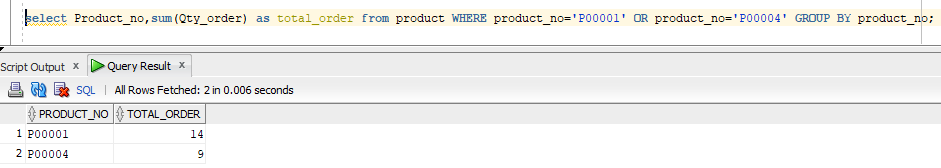
**Output:**

****

**Query-3: Retrieve the product no and the total quantity ordered for product’s ‘P00001’ and ‘P00004’ from product table.**

**SQL Statement:** select Product\_no,sum(Qty\_order) as total\_order from product WHERE product\_no='P00001' OR product\_no='P00004' GROUP BY product\_no;

**Output:**

****

**Query-4: Insert the following values into emp\_company.**

**SQL Statement:**

CREATE TABLE emp\_company (ENAME Varchar(10),CNAME Varchar(10),SALARY INT);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Anil' ,'ACC',1500);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Shankar' ,'TATA',2000);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Jay' ,'WIPRO',1800);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Sunil' ,'WIPRO',1700);

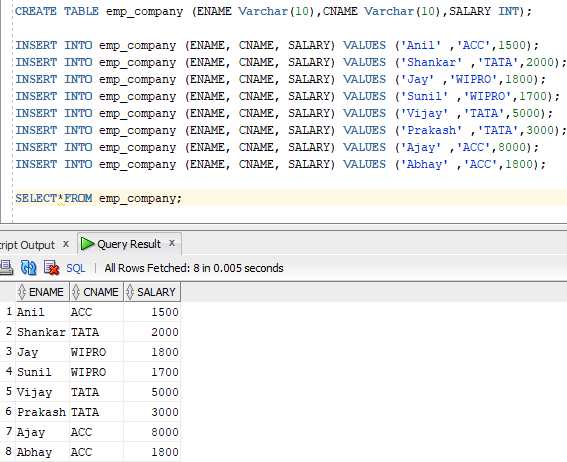
INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Vijay' ,'TATA',5000);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Prakash' ,'TATA',3000);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Ajay' ,'ACC',8000);

INSERT INTO emp\_company (ENAME, CNAME, SALARY) VALUES ('Abhay' ,'ACC',1800);

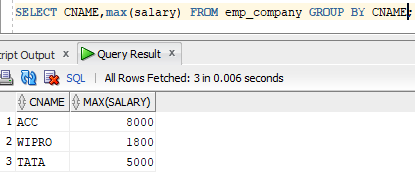
**Output:**

****

**Query-5: List the name of company and maximum salary in that company.**

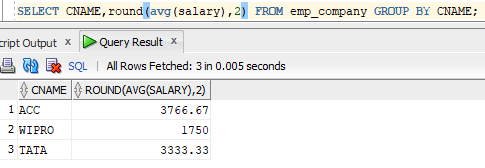
**SQL Statement:** SELECT CNAME,max(salary) FROM emp\_company GROUP BY CNAME;

**Output:**

****

**Query-6: Find out the average salary of each company.**

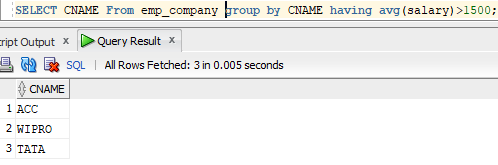
**SQL Statement:** SELECT CNAME,round(avg(salary),2) FROM emp\_company GROUP BY CNAME;

**Output:   
**

**Query-7: Find out the name of companies having average salary more than 1500.**

**SQL Statement:** SELECT CNAME From emp\_company group by CNAME having avg(salary)>1500;

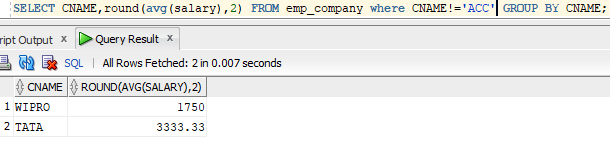
**Output:**

****

**Query-8: Find out the average salary of each company except ‘ACC’.**

**SQL Statement:** SELECT CNAME,round(avg(salary),2) FROM emp\_company where CNAME!='ACC' GROUP BY CNAME;

**Output:**

****

**// [IF any Questions are given to you]**

**Question-1:**

**Answer:**

**Question-2:**

**Answer:**

**PRACTICAL -7**

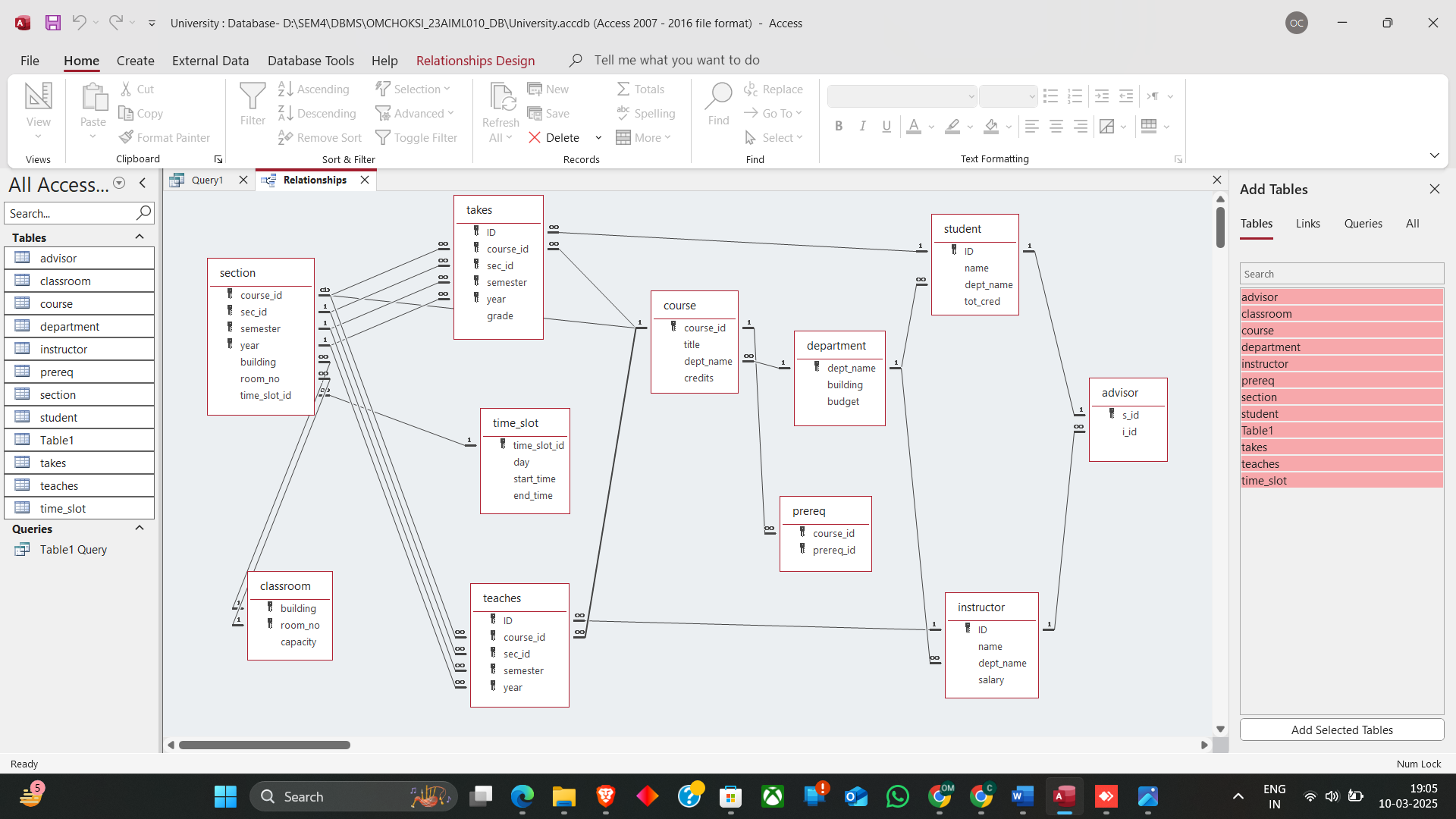
**ID: 23AIML070**

**Aim:** For a given University Schema, create tables and generate Master-Slave relationship along with all the necessary integrity constraints in MS Access Tool.

**GOAL IS TO MAKE THIS TYPE OF ENTITIY RELATIONSHIP SCHEMA FOR UNIVERSITY**

**A diagram of a diagram

AI-generated content may be incorrect.**

**OUTPUT:-** 

**PRACTICAL 8**

**AIM** - To apply the concept of integrity/data constraints while creating or altering tables in a database for managing sales-related data. FOR BELOW TASKS.

**1) Create Table Salespeople where Snum number(4) P.K, Sname varchar2(20)**

**Constraints**

1. **Primary Key Constraint**: Ensure Snum uniquely identifies each record. 2. **Not Null Constraint**: Ensure Sname is not null.

**Tasks:**

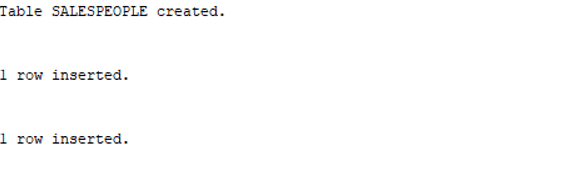
**Test Case 1: Insert a valid record.**

**Objective**: Verify that a valid record can be inserted into the Salespeople table. **Expected Result**: The record should be inserted successfully.

**Query:**



**Output:**



**Test Case 2: Insert a record with a duplicate Snum**

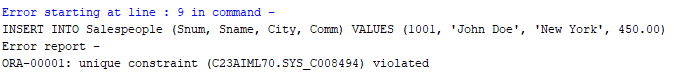
**Objective**: Verify that inserting a record with a duplicate Snum results in an error.

**Expected Result**: The insertion should fail with a primary key constraint violation error.

**Query:**



**Output:**



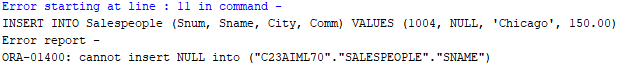
**Test Case 3: Insert a record with a null Sname**

**Objective**: Verify that inserting a record with a null Sname results in an error.

**Expected Result**: The insertion should fail with a not-null constraint violation error.

**Query:**

****

**Output: **

**Test Case 4: Insert a record with missing City and Comm**

**Objective**: Verify that a record can be inserted with City and Comm as null.

**Expected Result**: The record should be inserted successfully with City and Comm as null.

**Query:**

****

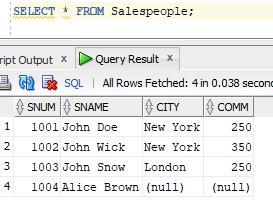
**Output:**



**Test Case 5: Retrieve all records from the Salespeople table.**

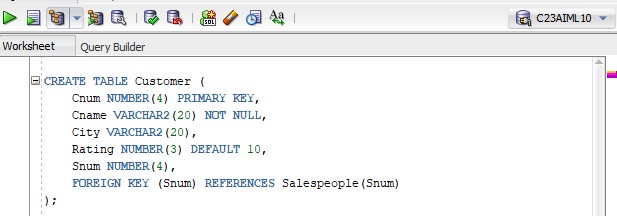
**Objective**: Verify that all records can be retrieved from the table.

**Query and output:**



**2) Create Table customer where Cnum number (4) P.K, Cname varchar2(20)NOT NULL, City varchar2(20), Rating number(3) DEFAULT 10, Snum number(4) F.K.(where snum refer salespeople table).**

**Query:**



**Output:**



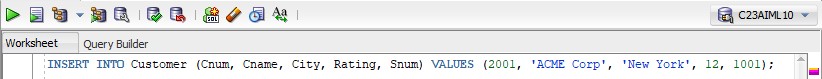
**Constraints**

1. **Primary Key Constraint**: Ensure Cnum uniquely identifies each record.
2. **Not Null Constraint**: Ensure Cname is not null.
3. **Foreign Key Constraint**: Ensure Snum references the Snum field in the Salespeople table. 4. **Default Constraint**: Ensure Rating defaults to 10 if no value is provided.

**Tasks:**

**Test Case 1**: Insert a valid record

**Objective**: Verify that a valid record can be inserted into the Customer table. **Expected Result**: The record should be inserted successfully. **Query:**



**Output:**



**Test Case 2**: Insert a record with a missing Rating

**Objective**: Verify that a record can be inserted with a Rating defaulting to 10.

**Expected Result**: The record should be inserted successfully with a Rating of 10.

**Query:**



**Output:**



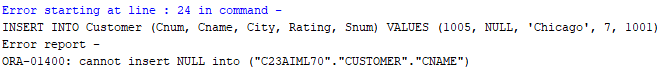
**Test Case 3**: Insert a record with a null Cname

**Objective**: Verify that inserting a record with a null Cname results in an error.

**Expected Result**: The insertion should fail with a not null constraint violation error.

**Query:** 

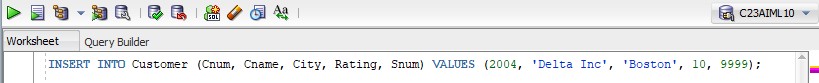
**Output:**



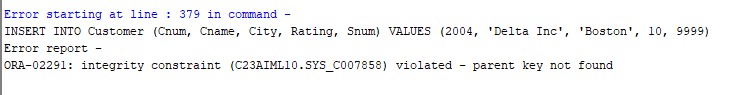
**Test Case 4**: Insert a record with a non-existent Snum

**Objective**: Verify that inserting a record with a non-existent Snum results in an error.

**Expected Result**: The insertion should fail with a foreign key constraint violation error. **Query:**



**Output:**



**Test Case 5**: Insert a record with missing City and Rating

**Objective**: Verify that a record can be inserted with City and Rating as null/default.

**Expected Result:** The record should be inserted successfully with City as null and Rating as 10.

**Query:**



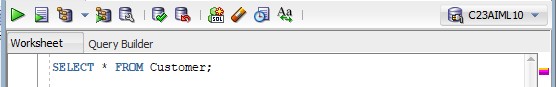
**Output:**



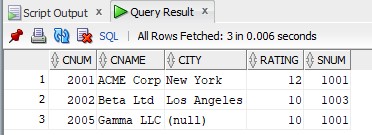
**Test Case 6**: Retrieve all records from the Customer table **Objective**: Verify that all records can be retrieved from the table.

**Expected Result:** The query should return all the inserted records.

**Query:**

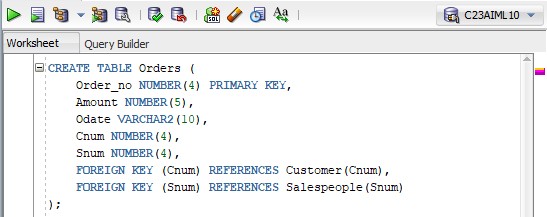


**Output:**



**3) Create table Order where Order\_no number(4) P.K, Amount number(5), Odate varchar2(10), Cnum number(4) F.K, (where cnum refer customer table). Snum number(4) F.K (where snum refers salespeople table).**

**Query:**



**Output:**



**Constraints**

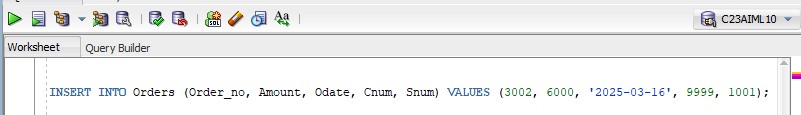
1. **Primary Key Constraint**: Ensure Order\_no uniquely identifies each record.
2. **Foreign Key Constraint**: Ensure Cnum references the Cnum field in the Customer table. 3. **Foreign Key Constraint**: Ensure Snum references the Snum field in the Salespeople table.

**Tasks:**

**Test Case 1**: Insert a valid record

**Objective**: Verify that a valid record can be inserted into the Orders table. **Expected Result**: The record should be inserted successfully.

**Query:**



**Output:**



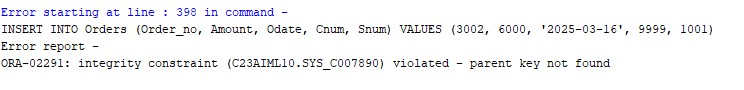
**Test Case 2**: Insert a record with a non-existent Cnum

**Objective**: Verify that inserting a record with a non-existent Cnum results in an error.

**Expected Result**: The insertion should fail with a foreign key constraint violation error for Cnum. **Query:**



**Output:**



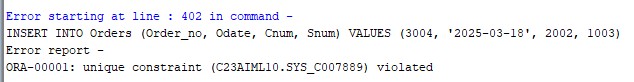
**Test Case 3**: Insert a record with a non-existent Snum

**Objective**: Verify that inserting a record with a non-existent Snum results in an error.

**Expected Result**: The insertion should fail with a foreign key constraint violation error for Snum. **Query:**



**Output:**



**Test Case 4**: Insert a record with a null Amount

**Objective**: Verify that a record can be inserted with Amount as null.

**Expected Result**: The record should be inserted successfully with Amount as null. **Query:**



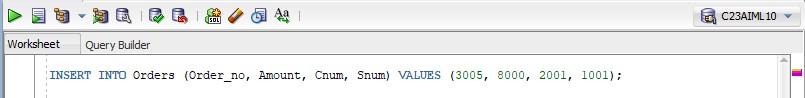
**Output:**



**Test Case 5**: Insert a record with null Odate

**Objective**: Verify that a record can be inserted with Odate as null.

**Expected Result:** The record should be inserted successfully with Odate as null. **Query:**



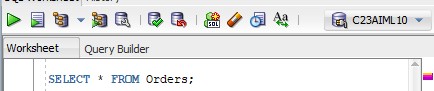
**Output:**



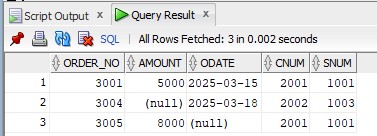
**Test Case 6:** Retrieve all records from the Orders table

**Objective**: Verify that all records can be retrieved from the table.

**Expected Result**: The query should return all the inserted records. **Query:**



**Output:**



# Table: Sales\_order

**Constraints**

∙ **Primary Key Constraints**: Ensure unique identification of records.

∙ **Foreign Key Constraints**: Maintain relationships between tables.

∙ **Not Null Constraints**: Ensure critical fields are not null.

∙ **Check Constraints**: Ensure data integrity by limiting column values.

∙ **Unique Constraints**: Ensure columns have unique values where required. ∙ **Default Constraints**: Assign default values to columns when no value is provided.

**Column Name Data Type Constraints :**

**Order\_no** Varchar2(6) Primary key/First letter starting with ‘O’

**Order\_date** Date

**Client\_no** Varchar2(6) Foreign Key references client\_no of client\_master table

**Dely\_addr** Varchar2(25)

**Salesman\_no** Varchar2(6) Foreign Key references sales\_no of

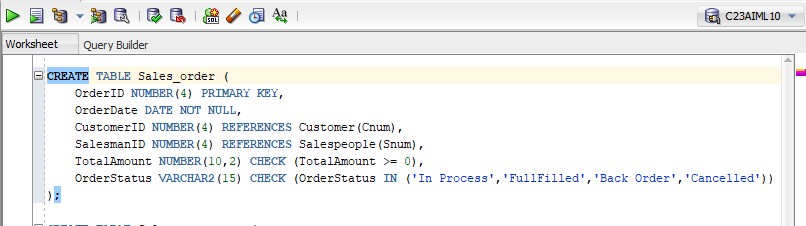
Salesman\_master table

**Dely\_type** char Delivery: part(P)/ full (f) Default ‘F’

**Order\_status** Varchar2 in Process, Fulfilled, Backorder, Cancelled

**Test Case**: Verify that all constraints, including the primary key, foreign key, check constraints, and default values, are correctly applied.

**Query:**



**Output:**



# Table: Salesman\_master

**Constraints**

∙ **Primary Key Constraints**: Ensure unique identification of records.

∙ **Foreign Key Constraints**: Maintain relationships between tables.

∙ **Not Null Constraints**: Ensure critical fields are not null.

∙ **Check Constraints**: Ensure data integrity by limiting column values.

∙ **Unique Constraints**: Ensure columns have unique values where required. ∙ **Default Constraints**: Assign default values to columns when no value is provided.

**Column name Data type Constraints :**

Salesman\_no Varchar2(6) The primary key/first letter must start with ‘S’

Salesman \_name Varchar2(20) Not null

Address Varchar2(30) Not null

City Varchar2(20)

Pincode Varchar2(8)

State Varchar2(20)

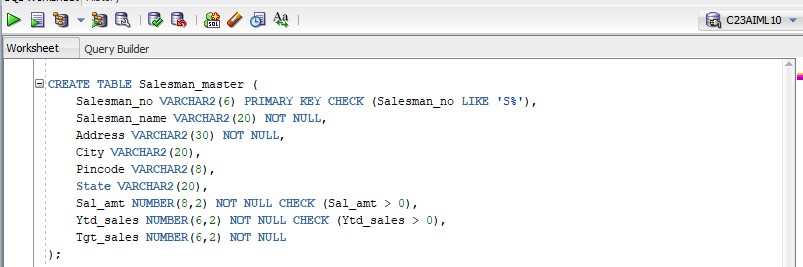
Sal\_amt Number(8,2) Not null, cannot be 0

Ytd\_sales Number(6,2) Not null, cannot be 0

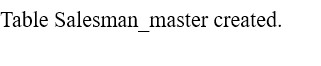
Tgt\_sales Number(6,2) Not null

**Test Case**: Verify that all constraints, including the primary key, not null, and check constraints, are correctly applied.

**Query:**



**Output:**



PAGE

**6. Table: Client\_master**

**Constraints**

∙ **Primary Key Constraints**: Ensure unique identification of records.

∙ **Foreign Key Constraints**: Maintain relationships between tables.

∙ **Not Null Constraints**: Ensure critical fields are not null.

∙ **Check Constraints**: Ensure data integrity by limiting column values.

∙ **Unique Constraints**: Ensure columns have unique values where required.

∙ **Default Constraints**: Assign default values to columns when no value is provided.

**Column name Data type Constraints**

Client\_no Varchar2(6) The primary key/first letter must start with ‘C’

Name Varchar2(20) Not null

Address Varchar2(30)

City Varchar2(15)

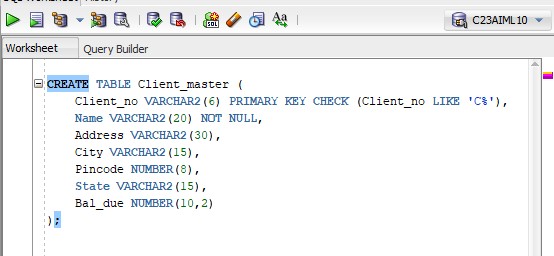
Pincode Number(8)

State Varchar2(15)

Bal\_due Number(10,2)

**Test Case**: Verify that all constraints, including the primary key, not null, and check constraints, are correctly applied.

**Query:**



**Output:**



**Practical-9**

**Aim:** To study and execute various JOIN commands to perform data retrieval and

manipulation from Salespeople, Customer, and Order tables based on specific

requirements.

**Theoratical Description:-**

Constraints:

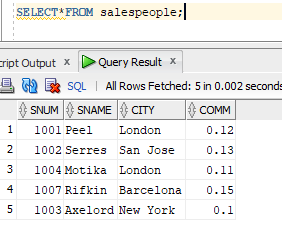
1. Primary Key - Ensures unique identification of records.
2. Foreign Key - Maintains referential integrity.
3. Not Null - Prevents essential fields from being empty.

Key Queries & Expected Results:

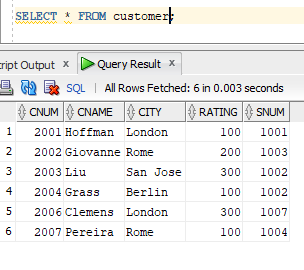
1. Retrieve customers serviced by specific salespeople.
2. Filter orders based on amount constraints.
3. Identify the largest orders for salespeople with high order values.
4. Match salespeople and customers based on city.
5. List orders with corresponding customer details.
6. Retrieve customers served by salespeople with high commission rates.
7. Compare customer ratings and find counts above the average.
8. Compute total sales per salesperson exceeding the highest single order.
9. Categorize customers into "High" or "Low" rating groups using UNION.

Different Tables are Given below

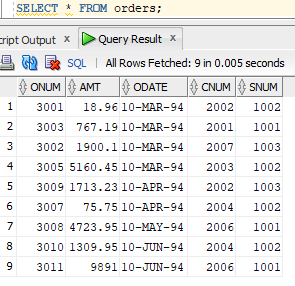
Salespeople:-



customer



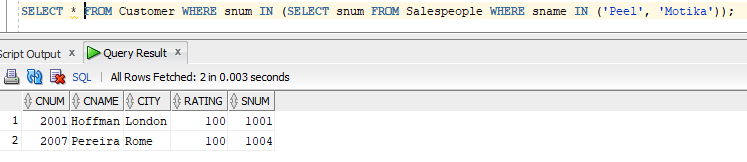
orders



**Query-1:** All customers serviced by Peel or Motika

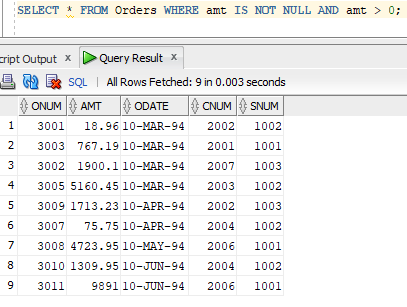
**SQL Statement:**SELECT \* FROM Customer WHERE snum IN (SELECT snum FROM Salespeople WHERE sname IN ('Peel', 'Motika'));

**Output:**



**Query-2:** All orders except those with 0 or null value in the amt field

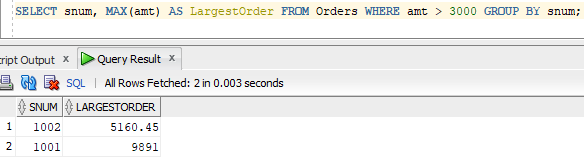
**SQL Statement:**SELECT \* FROM Orders WHERE amt IS NOT NULL AND amt > 0;

 **Output:**

**Query-3:** Largest order taken by each sales order value of more than 3000

**SQL Statement:**SELECT snum, MAX(amt) AS LargestOrder FROM Orders WHERE amt > 3000 GROUP BY snum;

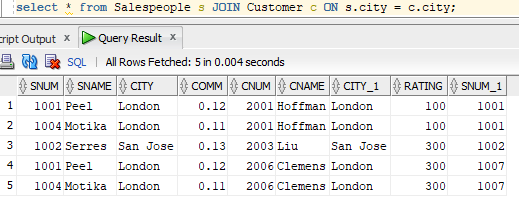
**Output:**



**Query-4:** All combinations of salespeople and customers who belong to the same city

**SQL Statement:**SELECT s.sname, c.cname, s.city FROM Salespeople s JOIN Customer c ON s.city = c.city;

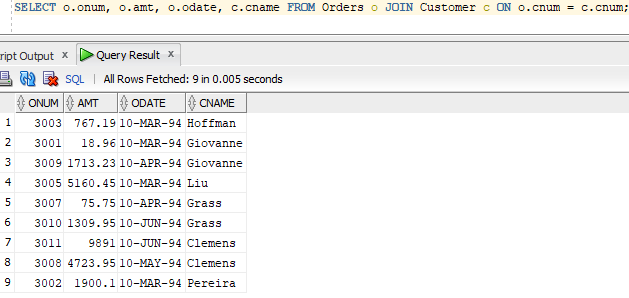
**Output:**



**Query-5:** List each order with the name of the customer who placed the order

**SQL Statement:**SELECT o.onum, o.amt, o.odate, c.cname FROM Orders o JOIN Customer c ON o.cnum = c.cnum;

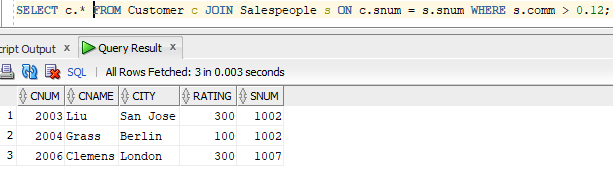
**Output:**



**Query-6:** Produce a listing of all the customers serviced by salespeople having a commission of more than 12%

**SQL Statement:**SELECT c.\* FROM Customer c JOIN Salespeople s ON c.snum = s.snum WHERE s.comm > 0.12;

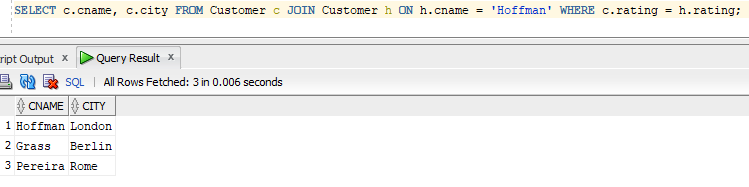
**Output:**



**Query-7:** Produce names and cities of all customers with the same rating as Hoffman

**SQL Statement:**SELECT c.cname, c.city FROM Customer c JOIN Customer h ON h.cname = 'Hoffman' WHERE c.rating = h.rating;

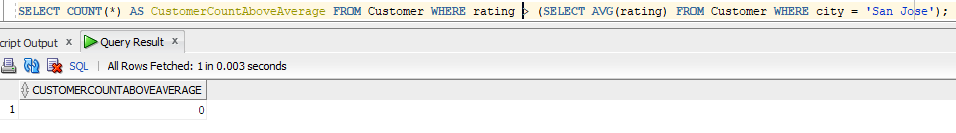
**Output:**



**Query-8:** Count the customers with ratings above San Jose’s average

**SQL Statement:**SELECT COUNT(\*) AS CustomerCountAboveAverage FROM Customer WHERE rating > (SELECT AVG(rating) FROM Customer WHERE city = 'San Jose');

**Output:**

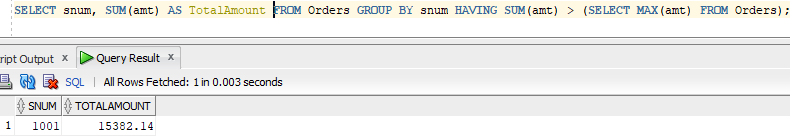


**Query-9:** Find the total amount in orders for each salesperson for whom this total is greater than the

amount of the largest order in the order table

**SQL Statement:**SELECT snum, SUM(amt) AS TotalAmount FROM Orders GROUP BY snum HAVING SUM(amt) > (SELECT MAX(amt) FROM Orders);

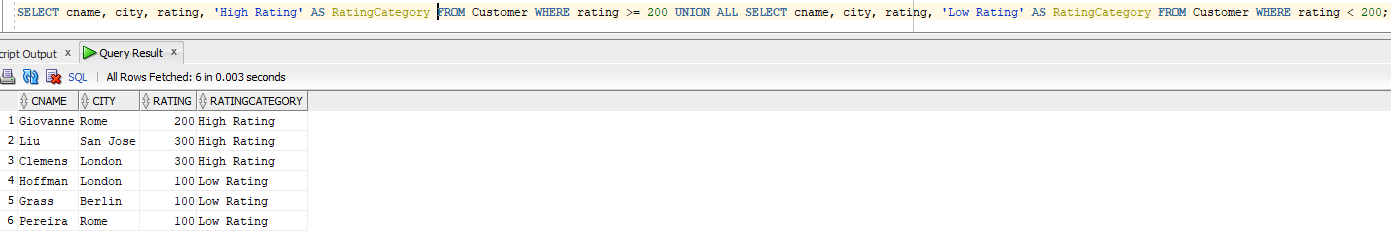
**Output:**



**Query-10:** Create a union of two queries that shows all customers' names, cities, and ratings. Those with a rating of 200 or greater will have the words ‘High Rating’ while others will have ‘Low Rating’.

**SQL Statement:**SELECT cname, city, rating, 'High Rating' AS RatingCategory FROM Customer WHERE rating >= 200 UNION ALL SELECT cname, city, rating, 'Low Rating' AS RatingCategory FROM Customer WHERE rating < 200;

**Output:**



Questions:-

**1)What is the purpose of using the JOIN clause in SQL queries?  
Answer:** The JOIN clause is used to combine rows from two or more tables based on a related column. For example, in Query 5, JOIN is used to retrieve customer names for each order by linking the NewOrder and NewCustomer tables through the cnum column.

**2)What is the difference between HAVING and WHERE in SQL?  
Answer:** WHERE filters rows before grouping, while HAVING filters groups after aggregation. For instance, in Query 9, HAVING is used to filter salespeople whose total order amount exceeds the largest single order.

**3)What is the role of the UNION operator in SQL?  
Answer:** The UNION operator is used to combine the results of two or more queries while ensuring unique records. In Query 10, UNION is used to classify customers into 'High Rating' and 'Low Rating' categories based on their rating.

# **PRACTICAL 10**

**AIM** - To create a PL/SQL procedure that performs the multiplication of two numbers.

**Theoretical Description :**

This PL/SQL procedure focuses on performing multiplication of two numeric input values while adhering to structured programming principles. It accepts two parameters of type NUMBER and provides the calculated result using an output parameter. The procedure also incorporates mechanisms to validate inputs and handle any errors effectively.

**Key Features and Constraints**

1. **Input Parameters**: The procedure takes two input parameters of data type NUMBER to ensure compatibility with numerical operations.
2. **Output Parameter**: The calculated product is returned via an output parameter for enhanced clarity in the code structure.
3. **Validation and Error Handling**: Comprehensive validation is implemented to address scenarios like incorrect data types or unexpected runtime issues, ensuring the procedure remains robust and reliable.

**Constraints**

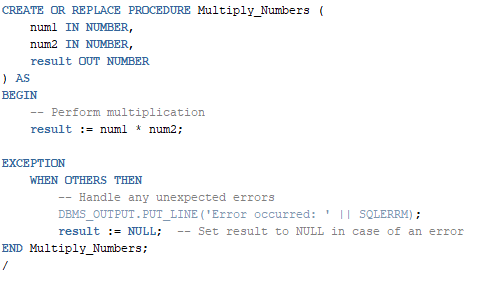
1. Input Parameters: The procedure will accept two input parameters of type NUMBER.

2. Output Parameter: The procedure will have an output parameter to return the result of the

multiplication.

3. Error Handling: The procedure will handle potential errors such as invalid input types.

**Query:**



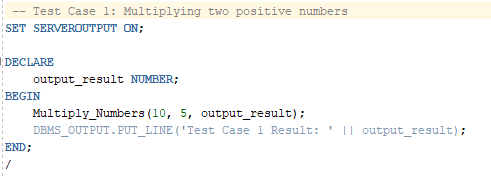
**Output:**

****

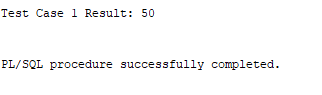
**Test Case 1**: Multiplying two positive numbers

**Input**: num1 = 10, num2 = 5

**Expected Output**: The result should be 50.



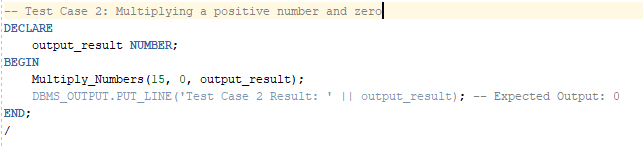
**Output** :



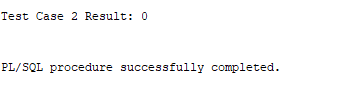
**Test Case 2**: Multiplying a positive number and zero

**Input**: num1 = 15, num2 = 0

**Expected Output**: The result should be 0.



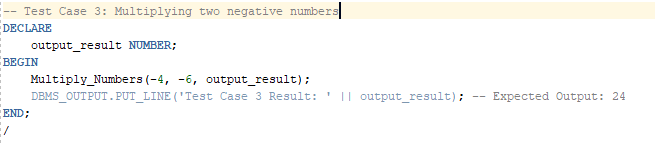
**Output** :



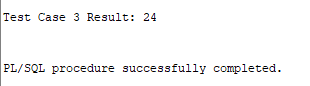
**Test Case 3**: Multiplying two negative numbers

**Input**: num1 = -4, num2 = -6

**Expected Output**: The result should be 24.



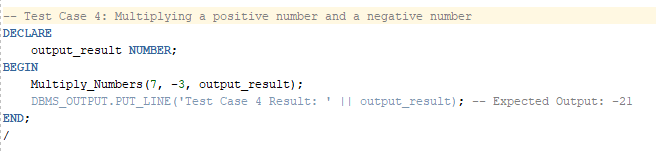
**Output** :



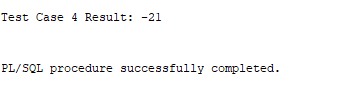
**Test Case 4**: Multiplying a positive number and a negative number

**Input**: num1 = 7, num2 = -3

**Expected Output**: The result should be -21.



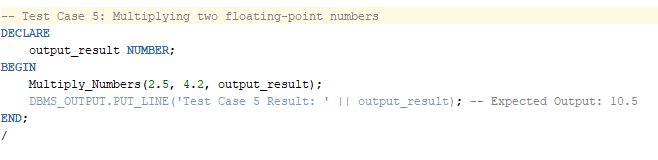
**Output** :



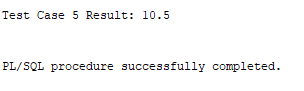
**Test Case 5**: Multiplying two floating-point numbers

**Input**: num1 = 2.5, num2 = 4.2

**Expected Output**: The result should be 10.5.



**Output** :



# **PRACTICAL 11**

**AIM** - **To create a PL/SQL block that deletes records from a table with age 21.**

**Additionally, it generates a trigger that stores the original record in another table**

**before deletion.**

**Constraints**

**1.** Deletion Condition: Delete records from the main table where the age is 21.

2. Trigger: Before deleting a record, store the original record in another table.

**SQL Query**

CREATE TABLE Persons (

person\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

age NUMBER

);

CREATE TABLE DeletedRecords (

person\_id NUMBER,

name VARCHAR2(50),

age NUMBER,

deletion\_date DATE

);

CREATE OR REPLACE TRIGGER trg\_store\_deleted

BEFORE DELETE ON Persons

FOR EACH ROW

BEGIN

INSERT INTO DeletedRecords (person\_id, name, age, deletion\_date)

VALUES (:OLD.person\_id, :OLD.name, :OLD.age, SYSDATE);

END;

/

INSERT INTO Persons VALUES (1, 'Alice', 21);

INSERT INTO Persons VALUES (2, 'Bob', 25);

INSERT INTO Persons VALUES (3, 'Charlie', 21);

INSERT INTO Persons VALUES (4, 'David', 30);

COMMIT;

DECLARE

v\_deleted\_count NUMBER;

BEGIN

DELETE FROM Persons

WHERE age = 21;

v\_deleted\_count := SQL%ROWCOUNT;

DBMS\_OUTPUT.PUT\_LINE('Number of records deleted: ' || v\_deleted\_count);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

ROLLBACK;

END;

/

SELECT \* FROM Persons;

SELECT \* FROM DeletedRecords;

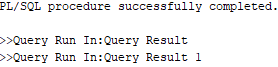
**Tasks -**

Test Case 1: Delete records where the age is 21

Objective: Verify that records with an age of 21 are successfully deleted.

Expected Result: Records with an age of 21 ('Alice' and 'Charlie') should be deleted.

Output:





Test Case 2: Verify Trigger Functionality

Objective: Verify that the trigger successfully stores original records in the DeletedRecords table before deletion

Expected Result: The DeletedRecords table should contain the original records of 'Alice' and 'Charlie' before they were deleted.

Output

