ASSIGNMENT 1

Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

1. SELECT query to retrieve all columns from a 'customers' table

```
SELECT * FROM CUSTOMERS;
```

CUSTOMER_ID	CUSTOMER_NAME	EMAIL	CITY
101	bhavesh	b@mail.com	Mumbai
102	ramesh	r@mail.com	Delhi
103	suresh	s@mail.com	Bengaluru
104	durgesh	d@mail.com	Mumbai
105	pratap	p@mail.com	Delhi

5 rows returned in 0.00 seconds CSV Export

2. Modify it to return only the customer name and email address for customers in a specific city.(FOR Mumbai)

```
SELECT CUSTOMER NAME, EMAIL
FROM CUSTOMERS
WHERE CITY='Mumbai';
```

CUSTOMER_NAME	EMAIL
bhavesh	b@mail.com
durgesh	d@mail.com

2 rows returned in 0.00 seconds CSV Export

ASSIGNMENT 2:

Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

1. INNER JOIN to Combine ORDERS and CUSTOMERS for Customers in a Specified Region

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME, c.EMAIL,
c.CITY, o.ORDER_ID, o.ORDER_NAME, o.REGION,
o.ORDER_VALUE
FROM CUSTOMERS c
INNER JOIN ORDERS o ON c.CUSTOMER_ID = o.CUSTOMER_ID
WHERE o.REGION = 'West';
```

CUSTOMER_ID	CUSTOMER_NAME	EMAIL	CITY	ORDER_ID	ORDER_NAME	REGION	ORDER_VALUE
101	bhavesh	b@mail.com	Mumbai	1	Order A	West	500
104	durgesh	d@mail.com	Mumbai	4	Order D	West	750

2 rows returned in 0.00 seconds CSV Export

2. LEFT JOIN to Display All Customers Including Those Without Orders

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME, c.EMAIL,
c.CITY, o.ORDER_ID, o.ORDER_NAME, o.REGION,
o.ORDER_VALUE
FROM CUSTOMERS c
LEFT JOIN ORDERS o ON c.CUSTOMER_ID = o.CUSTOMER_ID;
```

CUSTOMER_ID	CUSTOMER_NAME	EMAIL	CITY	ORDER_ID	ORDER_NAME	REGION	ORDER_VALUE
101	bhavesh	b@mail.com	Mumbai	1	Order A	West	500
102	ramesh	r@mail.com	Delhi	2	Order B	North	1500
103	suresh	s@mail.com	Bengaluru	3	Order C	South	2000
104	durgesh	d@mail.com	Mumbai	4	Order D	West	750
105	pratap	p@mail.com	Delhi	5	Order E	North	1250
106	vijay	v@mail.com	Kolkata	-	-	-	-

6 rows returned in 0.00 seconds CSV Export

ASSIGNMENT 3:

Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

1. Subquery to Find Customers Who Have Placed Orders Above the Average Order Value

```
SELECT c.CUSTOMER_ID, c.CUSTOMER_NAME, o.ORDER_ID,
o.ORDER_NAME, o.ORDER_VALUE
FROM CUSTOMERS c
JOIN ORDERS o ON c.CUSTOMER_ID = o.CUSTOMER_ID
WHERE o.ORDER_VALUE > (
    SELECT AVG(ORDER_VALUE) FROM ORDERS
);
```

CUSTOMER_ID	CUSTOMER_NAME	ORDER_ID	ORDER_NAME	ORDER_VALUE
102	ramesh	2	Order B	1500
103	suresh	3	Order C	2000
105	pratap	5	Order E	1250

3 rows returned in 0.00 seconds CSV Export

2. UNION Query to Combine Two SELECT Statements with the Same Number of Columns

```
SELECT CUSTOMER_ID, CUSTOMER_NAME, CITY
FROM CUSTOMERS
WHERE CITY = 'Mumbai'

UNION
```

```
SELECT CUSTOMER_ID, CUSTOMER_NAME, CITY
FROM CUSTOMERS
WHERE CITY = 'Delhi';
```

CUSTOMER_ID	CUSTOMER_NAME	CITY
101	bhavesh	Mumbai
102	ramesh	Delhi
104	durgesh	Mumbai
105	pratap	Delhi

4 rows returned in 0.00 seconds CSV Export

ASSIGNMENT 4:

Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

1. Begin a Transaction, Insert into ORDERS, and Commit the Transaction

```
BEGIN

INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_NAME, REGION, ORDER_VALUE) VALUES (6, 101, 'Order F', 'East', 1000);

COMMIT;

END;
```

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000

6 rows returned in 0.00 seconds

2. Begin a New Transaction, Update PRODUCTS, and Rollback the Transaction

```
BEGIN

UPDATE PRODUCTS
SET STOCK = STOCK - 10
WHERE PRODUCT_ID = 1;

ROLLBACK;

END;
```

Before rollback,

PRODUCT_ID	PRODUCT_NAME	STOCK	ORDER_ID
1	Product A	90	1
2	Product B	200	2
3	Product C	300	3

3 rows returned in 0.00 seconds

After rollback


```
Statement processed.
```

0.00 seconds

PRODUCT_ID	PRODUCT_NAME	STOCK	ORDER_ID
1	Product A	100	1
2	Product B	200	2
3	Product C	300	3

3 rows returned in 0.01 seconds

ASSIGNMENT 5:

Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

Step-by-Step Execution

1. First INSERT and SAVEPOINT:

```
BEGIN
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_NAME,
REGION, ORDER_VALUE) VALUES       (7, 102, 'Order G', 'South',
1100);
    SAVEPOINT sp1;
END;
```

Results Explain Describe Saved SQL History

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100

2. Second INSERT and SAVEPOINT:

```
BEGIN
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_NAME,
REGION, ORDER_VALUE) VALUES (8, 103, 'Order H', 'West', 1200);
   SAVEPOINT sp2;
END;
```

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100
8	103	Order H	West	1200

8 rows returned in 0.00 seconds

3. Third INSERT and SAVEPOINT:

```
BEGIN
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_NAME,
REGION, ORDER_VALUE) VALUES (9, 104, 'Order I', 'North', 1300);
   SAVEPOINT sp3;
END;
```

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100
8	103	Order H	West	1200
9	104	Order I	North	1300

9 rows returned in 0.00 seconds

4. Fourth INSERT and SAVEPOINT:

```
BEGIN
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_NAME,
REGION, ORDER_VALUE) VALUES (10, 105, 'Order J', 'East', 1400);
   SAVEPOINT sp4;
END;
```

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100
8	103	Order H	West	1200
9	104	Order I	North	1300
10	105	Order J	East	1400

10 rows returned in 0.00 seconds

5. Rollback to the second SAVEPOINT:

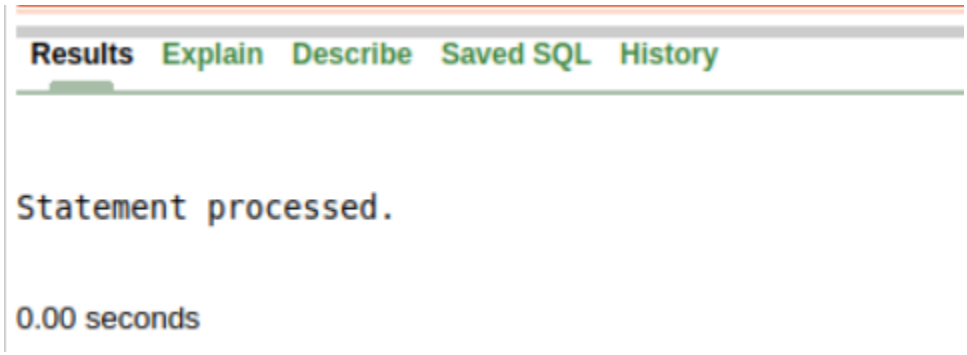
```
BEGIN
ROLLBACK TO SAVEPOINT sp2;
END;
```

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100
8	103	Order H	West	1200

8 rows returned in 0.00 seconds

6. Commit the overall transaction:

```
COMMIT;
```



Result:

ORDER_ID	CUSTOMER_ID	ORDER_NAME	REGION	ORDER_VALUE
1	101	Order A	West	500
2	102	Order B	North	1500
3	103	Order C	South	2000
4	104	Order D	West	750
5	105	Order E	North	1250
6	101	Order F	East	1000
7	102	Order G	South	1100
8	103	Order H	West	1200

8 rows returned in 0.00 seconds

After executing this script:
- The first two inserts (Order G and Order H) will be committed.
- The third and fourth inserts (Order I and Order J) will be rolled back and not committed.

ASSIGNMENT 6:

Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Title:
Use of Transaction Logs for Data Recovery: An Informative Overview

Introduction:
Transaction logs are like the unsung heroes of databases, silently safeguarding data integrity and aiding in data recovery during unforeseen system hiccups. In this report, we explore the pivotal role transaction logs play in restoring data after an unexpected shutdown, accompanied by a hypothetical scenario to bring the concept to life.

Importance of Transaction Logs:
Transaction logs are the recordkeepers of databases. They diligently jot down every change made, ensuring no data alteration goes unnoticed. By keeping a detailed record of these modifications, transaction logs become the backbone of data recovery, offering a reliable pathway to restore databases to a consistent state post-crash.

Scenario:
Imagine a bustling retail empire relying on its database to keep track of inventory, orders, and customer transactions. Suddenly, chaos ensues as the database server grinds to a halt due to an unforeseen power outage. With the database inaccessible, the risk of data loss looms large.

Utilization of Transaction Logs:
In the aftermath of the shutdown, the database management system (DBMS) springs into action, turning to transaction logs as its guiding light. These logs hold the key to unraveling the mystery of the database's last known consistent state before the abrupt shutdown.

Data Recovery Process:
1. Identification of Last Consistent State: The DBMS sifts through the transaction logs, sleuthing for clues to pinpoint the precise moment the database last stood on solid ground.

2. Replay of Transactions: Armed with this knowledge, the DBMS embarks on a journey to replay the recorded transactions. Like a meticulous conductor, it orchestrates the reintroduction of committed changes back into the database.

3. Rollback of Uncommitted Transactions: Yet, not all tales have happy endings. Any transactions left hanging in limbo at the time of the shutdown are gently ushered into oblivion, ensuring data integrity remains unscathed.

4. Verification and Recovery: With all transactions faithfully replayed, the DBMS conducts a thorough integrity check. Any discrepancies are promptly addressed, ensuring the database emerges from its ordeal unscathed and fully operational.

Conclusion:
Transaction logs might seem like the unsung heroes of databases, but their job in data recovery is truly remarkable. With their meticulous record-keeping and unwavering reliability, transaction logs act as the defenders of data integrity, guiding databases through the rough waters of system failures. The hypothetical situation we discussed highlights just how vital transaction logs are, giving us a peek into their crucial role in ensuring data stays safe and businesses keep running smoothly.