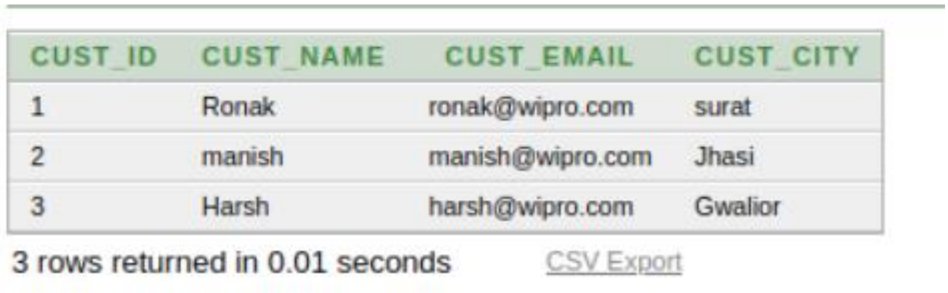
RDBMS and SQL

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.

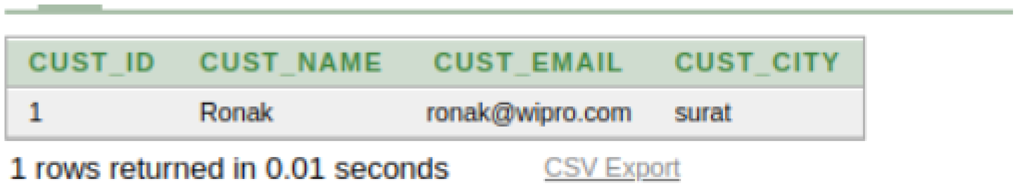
```
CREATE TABLE customers (Cust_Id number(3) Primary key, Cust_Name varchar(15) not null, Cust_Email varchar(20) not null, Cust_City varchar(20));

insert into customers values(1, 'Ronak', 'ronak@wipro.com','surat');

insert into customers values(2, 'manish', 'manish@wipro.com','Jhasi');

insert into customers values(3, 'Harsh', 'harsh@wipro.com',’Gwalior');

desc customers;

Select * from customers;
```



CUST_ID	CUST_NAME	CUST_EMAIL	CUST_CITY
1	Ronak	ronak@wipro.com	surat
2	manish	manish@wipro.com	Jhasi
3	Harsh	harsh@wipro.com	Gwalior

3 rows returned in 0.01 seconds CSV Export

```
Select Cust_Name, Cust_Email From customers where Cust_City = ‘surat’;
```



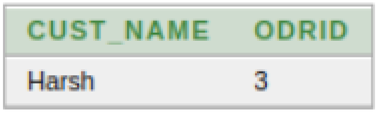
CUST_ID	CUST_NAME	CUST_EMAIL	CUST_CITY
1	Ronak	ronak@wipro.com	surat

1 rows returned in 0.01 seconds CSV Export

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.

```
SELECT customers.Cust_Name, orders.OdrID

FROM customers

INNER JOIN orders

ON customers.Cust_ID = orders.OdrId
```

WHERE Cust_City = 'Gwalior';



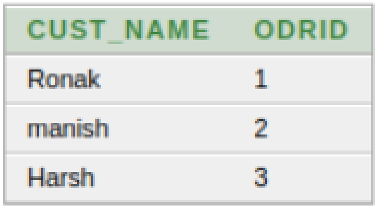
CUST_NAME	ODRID
Harsh	3

SELECT customers.Cust_Name, orders.OdrID

FROM customers

LEFT JOIN orders

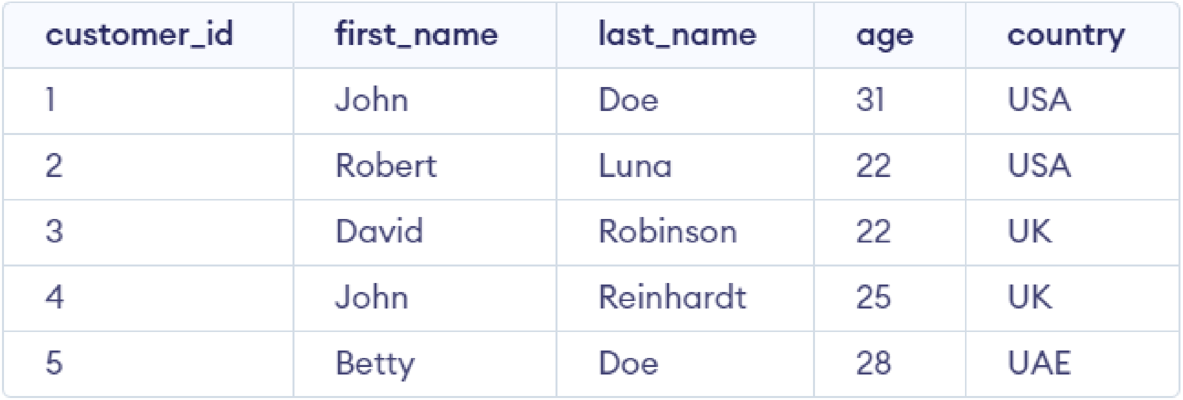
ON customers.Cust_ID = orders.OdrID;



CUST_NAME	ODRID
Ronak	1
manish	2
Harsh	3

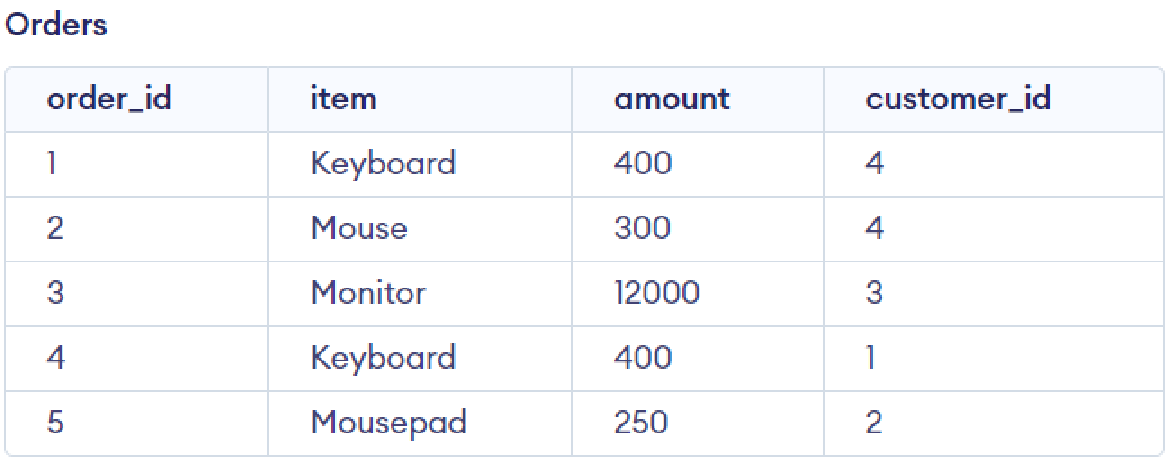
3. Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

Select * from Customers;



customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

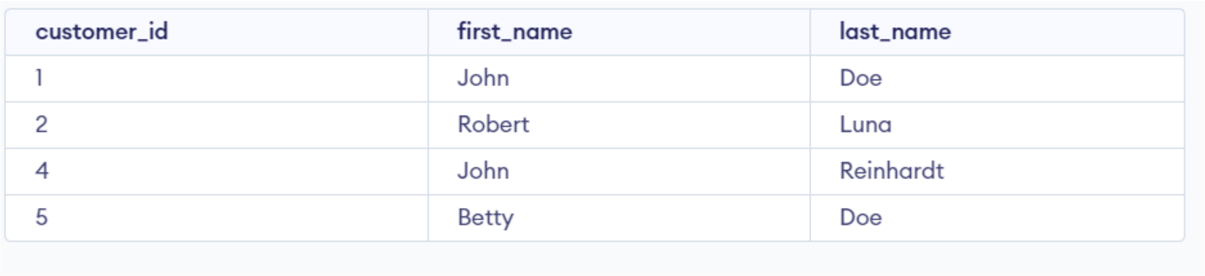
Select * from Orders;



Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT c.customer_id, c.first_name, c.last_name
FROM Customers c
WHERE c.customer_id NOT IN (
    SELECT o.customer_id
    FROM Orders o
    GROUP BY o.customer_id
    HAVING AVG(o.amount) > (
        SELECT AVG(amount) FROM Orders
    )
);
```

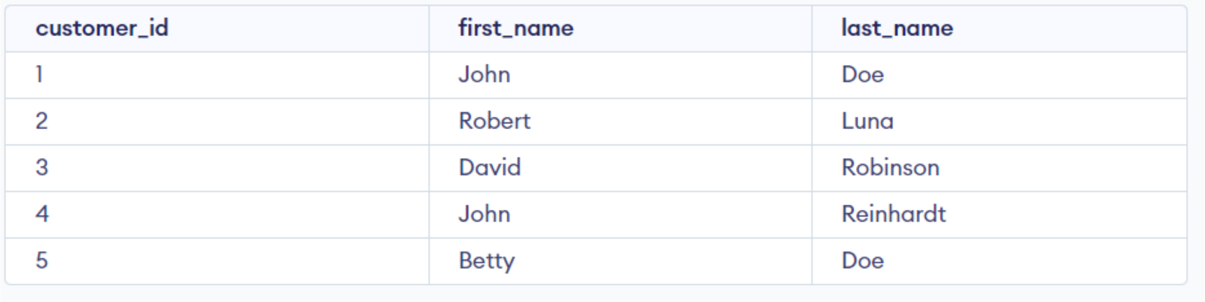


customer_id	first_name	last_name
1	John	Doe
2	Robert	Luna
4	John	Reinhardt
5	Betty	Doe

This query first calculates the average order value across all orders. It then uses a subquery to find the customer IDs of customers who have an average order value greater than the overall average. Finally, it selects the customer ID, first name, and last name for those customers.

```
SELECT c.customer_id, c.first_name, c.last_name
```

```
FROM Customers c
WHERE c.customer_id IN (
    SELECT o.customer_id
    FROM Orders o
    GROUP BY o.customer_id
    HAVING AVG(o.amount) > (
        SELECT AVG(amount) FROM Orders
    )
)
UNION
SELECT c.customer_id, c.first_name, c.last_name
FROM Customers c
WHERE c.customer_id NOT IN (
    SELECT o.customer_id
    FROM Orders o
    GROUP BY o.customer_id
    HAVING AVG(o.amount) > (
        SELECT AVG(amount) FROM Orders
    )
);
```



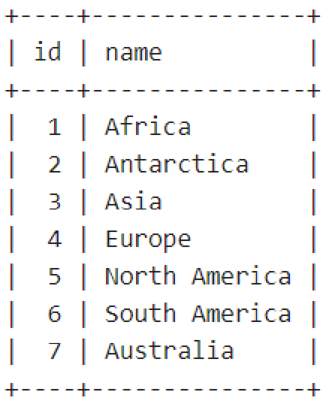
customer_id	first_name	last_name
1	John	Doe
2	Robert	Luna
3	David	Robinson
4	John	Reinhardt
5	Betty	Doe

This query first finds the customers with orders above the average order value using the same subquery as before. It then uses a UNION to combine those results with the customers who have orders at or below the average order value (i.e., the customers not in the first subquery). The result is a single result set containing all customers, regardless of their average order value.

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.

the SQL queries to begin a transaction, insert a new record into the 'continents' table, commit the transaction, update the 'continents' table, and rollback the transaction:

```
CREATE TABLE continents (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);


INSERT INTO continents (name) VALUES
('Africa'),
('Antarctica'),
('Asia'),
('Europe'),
('North America'),
('South America'),
('Australia');


select * from continents;
```

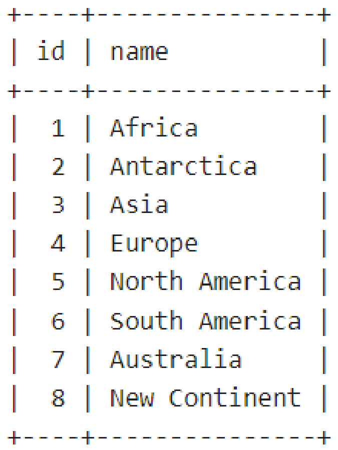


```
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Africa        |
|  2 | Antarctica    |
|  3 | Asia          |
|  4 | Europe        |
|  5 | North America |
|  6 | South America |
|  7 | Australia     |
+----+---------------+
```

→ **Begin** a transaction, **insert** a new record into the 'continents' table and **commit** the transaction.

```
START TRANSACTION;
INSERT INTO continents (name) VALUES ('New Continent');
COMMIT;
```

select * from continents;



```
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Africa        |
|  2 | Antarctica    |
|  3 | Asia          |
|  4 | Europe        |
|  5 | North America |
|  6 | South America |
|  7 | Australia     |
|  8 | New Continent |
+----+---------------+
```

→ **Update** the 'continents' table and **rollback** the transaction

START TRANSACTION;

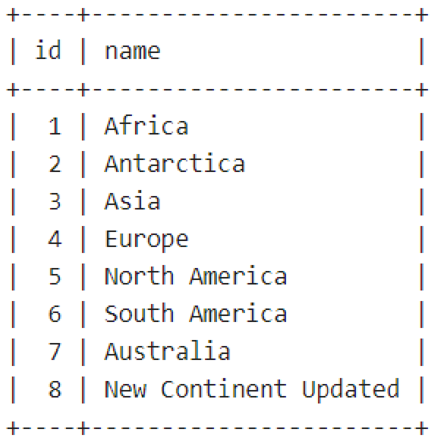
UPDATE continents

SET name = 'New Continent Updated'

WHERE name = 'New Continent';

select * from continents;

before rollback

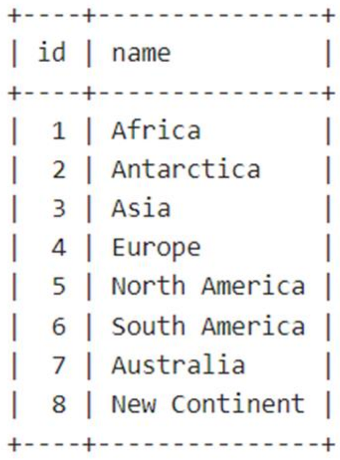


```
+----+-----------------------+
| id | name                  |
+----+-----------------------+
|  1 | Africa                |
|  2 | Antarctica            |
|  3 | Asia                  |
|  4 | Europe                |
|  5 | North America         |
|  6 | South America         |
|  7 | Australia             |
|  8 | New Continent Updated |
+----+-----------------------+
```

ROLLBACK;

select * from continents;

after rollback



```
+----+---------------+
| id | name          |
+----+---------------+
|  1 | Africa        |
|  2 | Antarctica    |
|  3 | Asia          |
|  4 | Europe        |
|  5 | North America |
|  6 | South America |
|  7 | Australia     |
|  8 | New Continent |
+----+---------------+
```

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.

creates a table named 'orders' and then performs a series of INSERTs into it, setting a SAVEPOINT after each, rolling back to the second SAVEPOINT, and committing the overall transaction:

```
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    item VARCHAR(50) NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    customer_id INT NOT NULL
);
```

→ Start a transaction

```
START TRANSACTION;
```

→ Save a point in the transaction

```
SAVEPOINT savepoint1;
```

→ Insert a new record into the 'orders' table

INSERT INTO orders (item, amount, customer_id) VALUES ('Item 1', 100.00, 1);

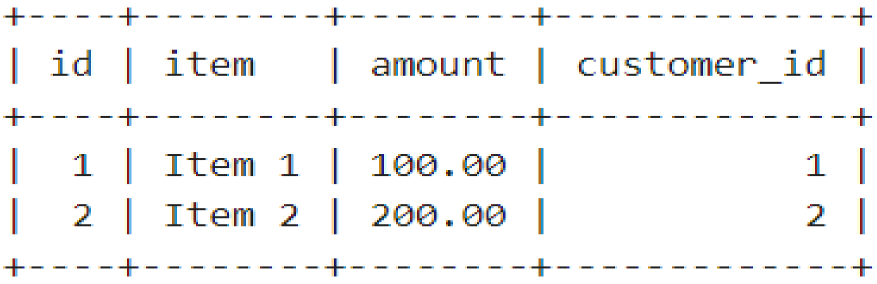
→ Save a point in the transaction

SAVEPOINT savepoint2;

→ Insert a new record into the 'orders' table

INSERT INTO orders (item, amount, customer_id) VALUES ('Item 2', 200.00, 2);

SELECT * FROM orders;



```
+----+--------+--------+-------------+
| id | item   | amount | customer_id |
+----+--------+--------+-------------+
|  1 | Item 1 | 100.00 |           1 |
|  2 | Item 2 | 200.00 |           2 |
+----+--------+--------+-------------+
```

→ Rollback to the second SAVEPOINT

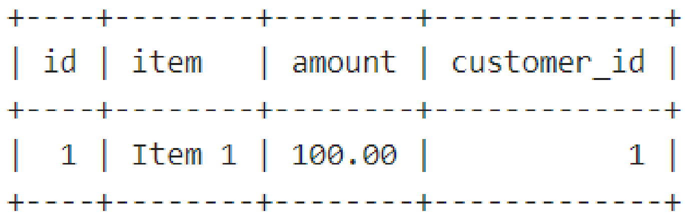
ROLLBACK TO SAVEPOINT savepoint2;

→ Commit the transaction

COMMIT;

→ Select all records from the 'orders' table

SELECT * FROM orders;



```
+----+--------+--------+-------------+
| id | item   | amount | customer_id |
+----+--------+--------+-------------+
|  1 | Item 1 | 100.00 |           1 |
+----+--------+--------+-------------+
```

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.

Brief report on the use of transaction logs for data recovery and a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown:

Use of Transaction Logs for Data Recovery

Transaction logs play a crucial role in data recovery in database management systems (DBMS). They record all the actions performed by transactions providing a reliable and consistent way to recover data in the event of unexpected shutdowns, hardware failures, or other system issues.

Hypothetical Scenario: Transaction Log Instrumental in Data Recovery

Let's consider a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Scenario

A company, Wipro, operates a customer relationship management (CRM) system to manage customer data. The CRM system is critical for the company's operations, and any data loss or corruption could have severe consequences.

Event

One day, there is an unexpected power outage that causes the CRM system to shut down abruptly. When the system is restarted, it fails to load properly, and some customer data appears to be missing or corrupted.

Data Recovery Process

To recover the lost or corrupted data, the system administrator initiates the data recovery process, leveraging the transaction log.

1. **Identifying the Last Consistent State:** The system administrator examines the transaction log to identify the last consistent state of the database before the unexpected shutdown. The transaction log contains a record of all the transactions that have been performed on the database, allowing the administrator to determine the point at which the data was in a consistent state.
2. **Undoing Changes with Log Records:** Using the log records stored in the transaction log, the system administrator begins the recovery process by undoing the changes

made by the failed transactions. The log records provide a detailed account of the modifications made, allowing the administrator to reverse the effects of the incomplete or failed transactions and restore the database to its previous state.

3. **Redoing Changes with Log Records:** After undoing the changes, the system administrator uses the log records stored in the transaction log to redo the changes made by the transactions that were in progress at the time of the unexpected shutdown. By reapplying the changes recorded in the log, the administrator ensures that the database reflects the modifications made by the transactions that were interrupted.
4. **Verifying Data Consistency:** Throughout the recovery process, the system verifies the consistency and validity of the log data. Checksums are used to ensure that the log data is consistent and has not been corrupted. If any invalid data is detected, appropriate actions are taken to prevent its use and ensure data integrity.

By following these steps and leveraging the transaction log, the system administrator successfully recovers the lost or corrupted data in the CRM system, restoring it to a consistent state.