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.

use testbd;

create table customer(cus_id int not null, cus_name varchar(20), cus_email varchar(30));

insert into customer values(101, ‘Shiva’, ‘shiva123@gmail.com’, ‘Pune’);

insert into customer values(102, ‘Ishvari, ‘ishvari@gmail.com’, ‘Mumbai’);

insert into customer values(103, ‘Prasad, ‘prasad321@gmail.com’, ‘Chennai’);

select * from customer;

select cus_name, cus_email from customer where city=’Pune’;

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.

SELECT c.customer_id, c.name, c.email, o.order_id, o.order_date FROM customers AS c INNER JOIN orders AS o ON c.customer_id = o.customer_id WHERE c.city = 'Pune';

SELECT c.customer_id, c.name, c.email, o.order_id, o.order_date FROM customers AS c LEFT JOIN orders AS o ON c.customer_id = o.customer_id;

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.

SELECT c.customer_id, c.name, c.email FROM customers AS c WHERE c.customer_id IN (SELECT customer_id FROM orders GROUP BY customer_id HAVING AVG(order_total) > (SELECT AVG(order_total) FROM orders));

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.

```
BEGIN TRANSACTION;

INSERT INTO orders (customer_id, order_date, total_amount) VALUES (10, CURRENT_TIMESTAMP, 199.99);

COMMIT;

UPDATE products SET stock_level = stock_level - 1 WHERE product_id = (SELECT product_id FROM inserted);

ROLLBACK;
```

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.

```
BEGIN TRANSACTION;

INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, CURRENT_TIMESTAMP, 100.00);

SAVEPOINT first_order;

INSERT INTO orders (customer_id, order_date, total_amount) VALUES (2, CURRENT_TIMESTAMP, 200.00);

SAVEPOINT second_order;

INSERT INTO orders (customer_id, order_date, total_amount) VALUES (3, CURRENT_TIMESTAMP, 300.00;

ROLLBACK TO SAVEPOINT second_order;

COMMIT;
```

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.

Transaction Logs: Guardians of Data Integrity

Transaction logs play a vital role in maintaining data integrity and ensuring data recovery in the event of unexpected system failures. They act as a chronological record of all database modifications, capturing every insert, update, and delete operation along with the time it was performed. This information becomes

invaluable for recovering data to a consistent state after issues like system crashes, power outages, or hardware malfunctions.

Functions of Transaction Logs:

- **Data Recovery:**
  - Transaction logs allow database administrators to roll forward or roll back incomplete transactions.
  - By analyzing the log entries for uncommitted transactions, they can identify the intended changes and either complete them (roll forward) or undo them (roll back) to ensure data consistency.
- **Improved Durability:**
  - Transaction logs provide a layer of assurance that data modifications are persisted to the database even if a system crash occurs during the write operation.
  - By replaying the log entries after a restart, the database can ensure all committed changes are reflected in the actual data.
- **Audit Trails and Compliance:**
  - Transaction logs serve as an audit trail, recording all database modifications and user activity.
  - This information can be crucial for regulatory compliance, forensic analysis, or identifying unauthorized access attempts.

Scenario: Recovering from the Unexpected

Imagine a busy online store running on a database system. During peak sales hours, a sudden power outage disrupts the system. The ongoing transactions, including several customer orders and inventory updates, are left in an uncertain state.

Here's where the transaction log comes to the rescue:

1. **System Restart:** After the power is restored, the database system restarts and recovers from the transaction log.
2. **Log Analysis:** The database administrator examines the log to identify all transactions in progress at the time of the shutdown.
3. **Uncommitted Transactions:** The log entries reveal several uncommitted transactions related to customer orders and inventory updates.
4. **Rollback or Rollforward:** Since the transactions weren't finalized, the administrator chooses to roll back these entries, effectively reversing any attempted changes.
5. **Data Consistency:** This ensures data consistency by preventing incomplete or potentially erroneous data from being incorporated into the database.

6. **Manual Intervention:** The administrator can then manually review the orders and reprocess them to minimize disruption for customers.