**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.

Query to retrieve all columns from the 'customers' table:

```sql
SELECT *

FROM customers;
```

query to return only the customer name and email address for customers in a specific city:

```sql
SELECT customer_name, email

FROM customers

WHERE city = 'New York';
```

**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.

```sql
SELECT customers.*, orders.*

FROM customers

INNER JOIN orders ON customers.customer_id = orders.customer_id

WHERE customers.region = 'West';

SELECT customers.*, orders.*

FROM customers

LEFT JOIN orders ON customers.customer_id = orders.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.

```sql
SELECT DISTINCT customers.*

FROM customers

INNER JOIN orders ON customers.customer_id = orders.customer_id

WHERE orders.order_value > (SELECT AVG(order_value) FROM orders);
```

```
SELECT customer_name, email

FROM customers

WHERE region = 'Region A'

UNION

SELECT customer_name, email

FROM customers

WHERE region = 'Region B';

SELECT customer_name, email

FROM customers

WHERE region = 'Region A'

UNION ALL

SELECT customer_name, email

FROM customers

WHERE region = 'Region B';
```

**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 (order_id, customer_id, order_date, order_value)

VALUES (1, 123, '2024-05-26', 500.00);


COMMIT;


BEGIN TRANSACTION;


UPDATE products
```

```
SET stock_quantity = stock_quantity - 10

WHERE product_id = 456;


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 (order_id, customer_id, order_date, order_value)

VALUES (1, 123, '2024-05-26', 500.00);

SAVEPOINT savepoint1;


INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (2, 124, '2024-05-26', 300.00);

SAVEPOINT savepoint2;


INSERT INTO orders (order_id, customer_id, order_date, order_value)

VALUES (3, 125, '2024-05-26', 450.00);

SAVEPOINT savepoint3;


ROLLBACK TO SAVEPOINT savepoint2;


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.

Report on the Use of Transaction Logs for Data Recovery

Introduction

Transaction logs are a critical component of database management systems (DBMS), serving as a record of all changes made to the database. They are essential for ensuring data integrity and enabling data recovery in the event of system failures, such as unexpected shutdowns, hardware malfunctions, or software crashes.

Purpose of Transaction Logs

Data Integrity: Transaction logs ensure that all database transactions are recorded in a sequential manner. This helps maintain the consistency and integrity of the database.

Crash Recovery: In the event of a system failure, transaction logs allow the DBMS to recover the database to a consistent state by replaying committed transactions and rolling back uncommitted ones.

Audit Trail: They provide a historical record of all transactions, which is useful for auditing and tracking changes made to the database over time.

Backup and Restore: Transaction logs are used in conjunction with backups to restore the database to a specific point in time, minimizing data loss.

Mechanism of Transaction Logs

Recording Transactions: Every transaction (INSERT, UPDATE, DELETE) is recorded in the transaction log before it is applied to the database. Each log entry includes details such as transaction ID, timestamp, affected rows, and the before and after state of the data.

Checkpointing: Periodically, the DBMS creates checkpoints that mark a point in the transaction log where the database is in a consistent state. Checkpoints help speed up the recovery process by providing a known good state from which to start applying log records.

Hypothetical Scenario: Data Recovery Using Transaction Logs

Scenario: Unexpected Shutdown

Imagine a retail company, "RetailCo," that uses a SQL-based DBMS to manage its inventory and sales data. During a busy holiday season, the database server experiences an unexpected power outage, causing an abrupt shutdown.

Steps for Data Recovery

Restart the Database Server:

The IT team restarts the database server after the power is restored.

Initiate Crash Recovery:

Upon startup, the DBMS automatically initiates the crash recovery process using the transaction log.

Identify the Last Checkpoint:

The DBMS locates the last checkpoint in the transaction log, which was created at 10:00 AM. At this point, the database was in a consistent state.

Apply Committed Transactions:

The DBMS scans the transaction log entries after the last checkpoint. It identifies and applies all transactions that were committed after 10:00 AM but before the shutdown. For example, an order placed at 10:05 AM is applied to the database.

Rollback Uncommitted Transactions:

The DBMS identifies transactions that were in progress but not committed at the time of the shutdown. These transactions are rolled back to ensure data consistency. For example, an inventory update that began at 10:15 AM but was not committed is undone.

Database Consistency:

By the end of the recovery process, the DBMS ensures that the database is brought back to a consistent state, reflecting all committed transactions up to the point of the failure and rolling back any incomplete transactions.

Conclusion

Transaction logs are indispensable for maintaining the reliability and integrity of databases. They enable effective data recovery by ensuring that all committed transactions are preserved and uncommitted ones are discarded during system failures. In the hypothetical scenario of "RetailCo," the transaction log plays a crucial role in restoring the database to its last consistent state after an unexpected shutdown, demonstrating the importance of this mechanism in real-world applications.