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.

Sol:

Below are the SQL statements to achieve the described operations:

1. Begin a transaction.
2. Insert a new record into the `orders` table.
3. Commit the transaction.
4. Begin a new transaction.
5. Update the `products` table.
6. Rollback the transaction.

SQL Statements

```
-- Begin the first transaction
BEGIN TRANSACTION;


-- Insert a new record into the 'orders' table
INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES
(101, 1, '2024-05-15', 250.00);


-- Commit the first transaction
COMMIT;


-- Begin the second transaction
BEGIN TRANSACTION;
-- Update the 'products' table
UPDATE products
SET stock_quantity = stock_quantity - 10
WHERE product_id = 5;
```

-- Rollback the second transaction

ROLLBACK;

Explanation

1. Begin the first transaction:

   BEGIN TRANSACTION;

   This starts a new transaction.

2. Insert a new record into the 'orders' table:

   INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

   (101, 1, '2024-05-15', 250.00);

   This statement inserts a new record into the `orders` table. Adjust the values as needed.

3. Commit the first transaction:

   COMMIT;

   This commits all the changes made in the current transaction, making them permanent.

4. Begin the second transaction:

   BEGIN TRANSACTION;

   This starts a new transaction.

5. Update the 'products' table:

   UPDATE products

   SET stock_quantity = stock_quantity - 10

   WHERE product_id = 5;

This statement updates the `products` table by decrementing the `stock_quantity` for a specific product. Adjust the `product_id` and the decrement value as needed.

6. Rollback the second transaction:

   ROLLBACK;

This undoes all the changes made in the current transaction, reverting the `products` table back to its previous state before the update.

These commands ensure that the first transaction (inserting into `orders`) is committed, while the second transaction (updating `products`) is rolled back.

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.

Sol:

Below are the SQL statements to:

1. Begin a transaction.

2. Perform a series of INSERTs into the `orders` table.

3. Set a SAVEPOINT after each INSERT.

4. Rollback to the second SAVEPOINT.

5. Commit the overall transaction.

SQL Statements

-- Begin the transaction

BEGIN TRANSACTION;

-- Insert first record and set the first savepoint

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(101, 1, '2024-05-15', 250.00);

SAVEPOINT savepoint1;

-- Insert second record and set the second savepoint

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(102, 2, '2024-05-16', 150.00);

SAVEPOINT savepoint2;

-- Insert third record and set the third savepoint

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(103, 3, '2024-05-17', 300.00);

SAVEPOINT savepoint3;

-- Insert fourth record and set the fourth savepoint

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(104, 4, '2024-05-18', 200.00);

SAVEPOINT savepoint4;

-- Rollback to the second savepoint

ROLLBACK TO SAVEPOINT savepoint2;

-- Commit the overall transaction

COMMIT;

Explanation

1. Begin the transaction:

   BEGIN TRANSACTION;

   This starts a new transaction.

2. Insert the first record and set the first savepoint:

   INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

   (101, 1, '2024-05-15', 250.00);

   SAVEPOINT savepoint1;

   This inserts a new record into the `orders` table and sets the first savepoint.

3. Insert the second record and set the second savepoint:

   INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

   (102, 2, '2024-05-16', 150.00);

   SAVEPOINT savepoint2;

   This inserts a second new record and sets the second savepoint.

4. Insert the third record and set the third savepoint:

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(103, 3, '2024-05-17', 300.00);

SAVEPOINT savepoint3;

This inserts a third new record and sets the third savepoint.

5. Insert the fourth record and set the fourth savepoint:

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES

(104, 4, '2024-05-18', 200.00);

SAVEPOINT savepoint4;

This inserts a fourth new record and sets the fourth savepoint.

6. Rollback to the second savepoint:

ROLLBACK TO SAVEPOINT savepoint2;

This rolls back the transaction to the state after the second savepoint, undoing the third and fourth inserts.

7. Commit the overall transaction:

COMMIT;

This commits all changes made in the transaction up to the second savepoint, making the first and second inserts permanent and discarding the third and fourth inserts.

==

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.

Sol:

Introduction

Transaction logs, also known as redo logs or write-ahead logs, are essential components of modern database management systems (DBMS). They record all changes made to the database, providing a reliable mechanism for data recovery in the event of system failures, crashes, or unexpected shutdowns. This report explores the use of transaction logs for data recovery and presents a hypothetical scenario illustrating their importance.

The Role of Transaction Logs in Data Recovery

Transaction logs serve several critical purposes in a DBMS:

1. Durability: Ensuring that once a transaction is committed, it remains so, even in the face of system failures.

2. Consistency: Helping to maintain the integrity of the database by ensuring that transactions are either fully completed or fully rolled back.

3. Data Recovery: Facilitating the restoration of the database to a consistent state after a crash or unexpected shutdown.

When a transaction is executed, changes are first written to the transaction log before being applied to the database. This process is known as write-ahead logging (WAL). In the event of a system failure, the DBMS can use the transaction log to redo completed transactions and undo incomplete ones, ensuring the database remains consistent and durable.

Hypothetical Scenario: Data Recovery After an Unexpected Shutdown

Scenario Description

Imagine a retail company, RetailCo, that uses a DBMS to manage its sales transactions. One day, RetailCo's database server experiences an unexpected shutdown due to a power outage. At the time of the shutdown, several transactions were in progress, and others had recently been committed.

Transactions Before Shutdown

- Transaction 1: Insert a new order into the `orders` table.

- Transaction 2: Update the stock quantity in the `products` table.

- Transaction 3: Commit a sale and update the customer’s purchase history.

Transaction Log Entries

1. Begin Transaction 1:

```
BEGIN TRANSACTION;
```

- Insert new order:

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES (201, 5, '2024-05-15', 120.00);

- Savepoint:

SAVEPOINT savepoint1;

2. Begin Transaction 2:

BEGIN TRANSACTION;

- Update stock quantity:

UPDATE products SET stock_quantity = stock_quantity - 3 WHERE product_id = 10;

- Savepoint:

SAVEPOINT savepoint2;

3. Begin Transaction 3:

BEGIN TRANSACTION;

- Update customer purchase history:

UPDATE customers SET purchase_history = '2024-05-15: Order 201' WHERE customer_id = 5;

-Savepoint:

SAVEPOINT savepoint3;

Steps for Data Recovery

1. Redo Committed Transactions: After the system restarts, the DBMS reads the transaction log to identify committed transactions. For example, if Transaction 1 was committed before the shutdown, the DBMS re-applies this transaction to ensure the new order is recorded.

INSERT INTO orders (order_id, customer_id, order_date, total_amount) VALUES (201, 5, '2024-05-15', 120.00);

2. Undo Uncommitted Transactions: Transactions that were in progress but not committed at the time of the shutdown are rolled back. For instance, if Transaction 2 and Transaction 3 were not committed, their changes are undone to maintain database consistency.

ROLLBACK TO SAVEPOINT savepoint2;

```
ROLLBACK TO SAVEPOINT savepoint3;
```

3. Restore Database to Consistent State: The transaction log ensures that the database is returned to its last consistent state by redoing committed transactions and undoing uncommitted ones. In this scenario, only the changes from committed transactions are applied, preserving data integrity.

Conclusion

Transaction logs are vital for ensuring data durability and consistency in a DBMS. They provide a robust mechanism for recovering data after unexpected system failures by maintaining a detailed record of all transactions. The hypothetical scenario of RetailCo illustrates how transaction logs can be instrumental in data recovery, allowing the company to resume operations with minimal data loss and ensuring the integrity of its database.

By implementing transaction logs and following proper recovery procedures, organizations can safeguard their data against unexpected disruptions and maintain business continuity.