**Database Transactions: Commit, Rollback, Savepoint**

A transaction is a sequence of operations performed as a single logical unit of work. In a database system, transactions ensure data integrity and consistency, especially in multi-user environments. Here are some practical notes on how transactions work with `COMMIT`, `ROLLBACK`, and `SAVEPOINT`.

---

**1. Transaction**

- A transaction is a group of operations that are executed together.

- The primary properties of transactions are defined by ACID principles:

- Atomicity: The entire transaction is treated as a single unit, and all operations must complete successfully, or none should.

- Consistency: The database must be in a consistent state before and after the transaction.

- Isolation: Transactions are executed independently without interference from other transactions.

- Durability: Once a transaction has been committed, the changes are permanent.

# Starting a Transaction:

Most databases automatically start a transaction for every set of queries until a `COMMIT` or `ROLLBACK` is executed. In manual control, you explicitly begin a transaction using commands like:

```sql

BEGIN TRANSACTION;

```

**2. Commit**

- The `COMMIT` command is used to save the changes made in the current transaction permanently.

- Once you issue a `COMMIT`, the changes are finalized, and the database moves to a new state.

- Example:

```sql

BEGIN TRANSACTION;

INSERT INTO employees (name, role) VALUES ('Alice', 'Engineer');

COMMIT;

```

After executing the `COMMIT` statement, the new employee "Alice" will be permanently added to the `employees` table.

**3. Rollback**

- The `ROLLBACK` command is used to undo changes made during a transaction. Any modifications made since the last `BEGIN TRANSACTION` will be reversed, and the database will return to its previous state.

- Example:

```sql

BEGIN TRANSACTION;

DELETE FROM employees WHERE name = 'Alice';

ROLLBACK;

```

If `ROLLBACK` is executed after the `DELETE` statement, the employee "Alice" will not be removed from the `employees` table, and the database will stay as it was before the transaction.

# Use Case:

- If an error occurs, or if you detect an issue during the execution of a transaction, you can use `ROLLBACK` to prevent invalid or partial changes.

**4. Savepoint**

- A `SAVEPOINT` allows you to set intermediate points within a transaction. You can ROLLBACK to a specific `SAVEPOINT` without affecting the entire transaction.

- This is useful when you want to test a particular operation within a larger transaction but want the flexibility to revert some changes without rolling back the entire set of changes.

- Example:

```sql

BEGIN TRANSACTION;

INSERT INTO employees (name, role) VALUES ('Bob', 'Manager');

SAVEPOINT savepoint\_1;

INSERT INTO employees (name, role) VALUES ('Carol', 'Designer');

-- If something goes wrong with Carol's insertion, rollback to savepoint\_1

ROLLBACK TO savepoint\_1;

COMMIT;

```

In this example, "Bob" will still be added, but the addition of "Carol" will be undone if we rollback to `savepoint\_1`.

# Points to Note:

- You can create multiple savepoints in a transaction, but a rollback to a savepoint doesn't affect savepoints created before it.

- You can also release a savepoint if it's no longer needed using:

```sql

RELEASE SAVEPOINT savepoint\_name;

```

---

**Practical Example: Transaction with Commit, Rollback, and Savepoint**

```sql

BEGIN TRANSACTION;

-- Insert data into employees

INSERT INTO employees (name, role) VALUES ('Alice', 'Engineer');

INSERT INTO employees (name, role) VALUES ('Bob', 'Manager');

-- Create a savepoint after inserting two rows

SAVEPOINT savepoint\_1;

-- Attempt to insert another row, but assume an error occurs

INSERT INTO employees (name, role) VALUES ('Carol', 'Designer');

-- Something goes wrong, rollback to savepoint\_1

ROLLBACK TO savepoint\_1;

-- Now insert the correct data and commit the transaction

INSERT INTO employees (name, role) VALUES ('David', 'Designer');

-- Finalize the transaction

COMMIT;

```

In this example:

- "Alice" and "Bob" are inserted successfully.

- We use `SAVEPOINT savepoint\_1` to mark the state after inserting Alice and Bob.

- After an issue with inserting "Carol," we rollback to `savepoint\_1`.

- Finally, "David" is inserted, and the transaction is committed.

---

**Best Practices for Transactions:**

1. Keep Transactions Short: The longer the transaction, the greater the chance of conflicts with other users or processes.

2. Error Handling: Always anticipate potential errors in transactions, especially when handling complex operations or multiple table updates.

3. Locking: Some transactions may lock resources (tables, rows). Be mindful of how long these locks persist to prevent database performance issues.

4. Use Savepoints Wisely: Savepoints are great for partial rollbacks but can add complexity, so use them when really needed.

In mysql where you start txn then rollback state reached starting state.