1. Transactions: COMMIT, ROLLBACK, and SAVEPOINT

Transactions are a fundamental concept in database management, ensuring data integrity and reliability. A transaction is a single logical unit of work that contains one or more SQL statements.

The key properties of a transaction are often referred to as **ACID**:

- **Atomicity:** All operations within a transaction are completed successfully, or none are. It's an "all or nothing" proposition.
- **Consistency:** A transaction brings the database from one valid state to another valid state.
- **Isolation:** Concurrent transactions do not interfere with each other. Each transaction appears to execute in isolation.
- **Durability:** Once a transaction is committed, its changes are permanent and survive system failures.

By default, MySQL runs in auto commit mode, meaning each SQL statement is treated as a separate transaction and committed immediately. For multi-step operations that must succeed or fail, you need to explicitly manage transactions.

1.1. START TRANSACTION or BEGIN

This command marks the beginning of a new transaction. All subsequent SQL statements (DML - INSERT, UPDATE, DELETE) will be part of this transaction until COMMIT or ROLLBACK is issued.

Syntax:

```
START TRANSACTION;
-- OR
BEGIN;
```

1.2. COMMIT

The COMMIT command is used to save all changes made during the current transaction permanently to the database. Once committed, the changes cannot be undone using ROLLBACK.

Syntax:

```
COMMIT.
```

Example: Imagine you need to transfer an employee from one department to another and update their job title, ensuring both changes happen together.

```
USE employee_management;
```

```sql
-- Start a new transaction
START TRANSACTION;

-- Update the employee's department (e.g., Bob from Engineering to Sales)
UPDATE Employees SET dept_id = 3 WHERE employee_id = 2; -- Bob Johnson

-- Update their job title
UPDATE Employees SET job_id = 'SALES_ENG' WHERE employee_id = 2;

-- Check the changes (these are not yet permanent in the database for others)
SELECT * FROM Employees WHERE employee_id = 2;

-- If everything looks correct, commit the changes

COMMIT;

-- Now the changes are permanently saved and visible to all users
SELECT * FROM Employees WHERE employee_id = 2;
```

1.3. ROLLBACK;

The ROLLBACK command is used to undo all changes made during the current transaction. This effectively reverts the database to the state it was in before START TRANSACTION; was issued. No changes from the transaction will be saved.

Syntax:

```sql
ROLLBACK;
```

Example: Suppose you start a transaction, make some changes, but then realize there's an error or you don't want to save them.

```sql
USE employee_management;

-- Start a new transaction
```

```
START TRANSACTION.

-- Attempt to increase salary for all sales employees by 10%
UPDATE Employees SET salary = salary * 1.10 WHERE dept_id = 3
-- Also try to delete an employee (mistake!)
DELETE FROM Employees WHERE employee_id = 1; -- Accidentally deleting Alice
-- Check the changes (these are not yet permanent)
SELECT * FROM Employees;

-- Oh no, I deleted Alice by mistake! Let's undo everything in this transaction.

ROLLBACK.

-- Verify that Alice is back and no salaries were updated
SELECT * FROM Employees;
```

1.4. SAVEPOINT; and ROLLBACK TO SAVEPOINT;

SAVEPOINT allows you to create named checkpoints within a transaction. This is useful when you want to partially undo a transaction, reverting only to a specific savepoint rather than the entire transaction.

- **SAVEPOINT savepoint_name;:** Creates a named savepoint.
- **ROLLBACK TO SAVEPOINT savepoint_name;:**
    - Undoes all changes made *after* the specified savepoint
    - Changes made *before* the savepoint are retained within the transaction.
- **RELEASE SAVEPOINT savepoint_name;:** Removes a savepoint. Note that savepoints are automatically removed when a transaction is committed or rolled back fully.

Syntax:

1. **SAVEPOINT savepoint_name;**
2. **ROLLBACK TO SAVEPOINT savepoint_point_name;**
3. **RELEASE SAVEPOINT savepoint_name;**

Example: Let's update multiple employee records in a transaction, but keep a rollback point just in case.

```
USE employee_management;

-- Start a new transaction
START TRANSACTION;

-- Update salary for all Engineering employees
UPDATE Employees SET salary = salary + 5000 WHERE dept_id = 2;
SELECT * FROM Employees WHERE dept_id = 2;

-- Create a savepoint here
SAVEPOINT after_eng_salary_update;

-- Now, attempt to update salary for all Marketing employees (but let's say this causes an issue)
UPDATE Employees SET salary = salary + 2000 WHERE dept_id = 4;
SELECT * FROM Employees WHERE dept_id = 4;

-- Oh, the Marketing salary update had an unexpected result or violated a business rule.
-- Let's rollback ONLY the changes made after 'after_eng_salary_update'.
ROLLBACK TO SAVEPOINT after_eng_salary_update;
As a result – not written to the database

-- Verify: Engineering salaries are still updated, but Marketing salaries are reverted
SELECT * FROM Employees WHERE dept_id = 2; -- Should show updated salary
SELECT * FROM Employees WHERE dept_id = 4; -- Should show original salary

-- The transaction is still active. You can make more changes or commit/rollback the whole thing.
-- Let's commit the Engineering salary update (the only change remaining in the transaction)

COMMIT;
-- Verify the final state
SELECT * FROM Employees;
```