TOPS TECHNOLOGY

Module 4 – Introduction to DBMS

Presented By :

Nandni Vala

Rollback and Commit Savepoint

1. Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?

- **ROLLBACK to SAVEPOINT**:
- When a **ROLLBACK** is issued to a specific savepoint, only the changes made after that savepoint are undone, and all operations before the savepoint remain committed.
- This allows partial rollback, meaning you can undo only a part of the transaction while keeping the rest intact.
- **Example**:
- SAVEPOINT save1;
- -- Perform some operations
- SAVEPOINT save2;
- -- Perform more operations
- ROLLBACK TO save1; -- Undo changes made after save1
- -- Changes made after save1 are rolled back, but the ones before save1 remain.

- COMMIT:
- A **COMMIT** permanently saves all the changes made in the transaction to the database. Once committed, the changes are final, and no further rollback is possible, including rolling back to savepoints.
- After a commit, savepoints are no longer valid, and any rollback to them will result in an error.
  - Example:
- SAVEPOINT save1;
- -- Perform some operations
- COMMIT; -- All changes are saved, and savepoints are no longer valid.
- **ROLLBACK (without specifying SAVEPOINT)**:
- A **ROLLBACK** without specifying a savepoint undoes all changes made during the entire transaction and resets the transaction to its initial state before any SQL operations were executed.

- **Example**:
- ROLLBACK; -- Undo all changes made during the entire transaction

2. When is it useful to use savepoints in a database transaction?

- Savepoints are useful in a database transaction when you need to:
- **Rollback part of a transaction**: Undo specific changes without affecting the entire transaction.
- **Manage complex transactions**: Break a long transaction into smaller steps and undo only the problematic part.
- **Improve error handling**: Gracefully handle errors by rolling back to a savepoint instead of the entire transaction.
- **Test and debug**: Simulate failures and test different scenarios without affecting the overall database state.
- **Minimize the impact of rollbacks**: Reduce the scope of a rollback for more efficient recovery.

2.When would you use an explicit cursor over an implicit one?

- You would use an **explicit cursor** over an **implicit cursor** when you need to process multiple rows individually, require fine control over the cursor (e.g., opening, fetching, and closing), or need to handle complex queries with custom logic.
- Explicit cursors are ideal for iterating over result sets, handling dynamic conditions, and managing multiple cursors in a single block of code.
- For simple SQL operations returning single rows or for automatic query handling, implicit cursors are sufficient.