**INMT202 Self Study**

**Chapter 10**

**So, What is Transaction in terms of Databases??**

In a database, a **transaction** is a group of operations (like reading, writing, updating data) that should be treated as one single task. Imagine it like a "package deal"—either everything inside the package happens correctly, or none of it happens at all.

**Why Transactions Matter in Databases**

1. **Ensuring Data Accuracy**: Think of a banking app where you transfer money. You don't want the money to leave your account but not reach the recipient's account, right? A transaction makes sure both actions (withdrawing and depositing) either complete successfully together or not at all.
2. **Dealing with Multiple Users**: In a database, many people or systems can try to access and modify the same data at the same time. Transactions ensure that this doesn’t lead to confusion or incorrect data.

**The Five Key Concepts of Transactions (ACID Properties)**

These rules (ACIDS) make sure transactions work safely:

1. **Atomicity** – All or nothing. Imagine you're buying something online. Your transaction will either succeed fully (money is deducted, item is ordered) or fail, and nothing changes.
2. **Consistency** – Keeps the database in a good state. After a transaction, the data should follow the rules (like not allowing negative balances).
3. **Isolation** – Each transaction happens separately from others. If two people are transferring money at the same time, their transactions won't mess with each other.
4. **Durability** – Once a transaction is done, the change is permanent. Even if the system crashes, the completed transaction remains.
5. **Serializability** – Ensures that the schedule for the concurrent execution of several transactions should yield consistent results.

**Example of a Transaction in a Database:**

Let’s say you have a simple database with customer account balances. Here’s a transaction:

1. **Step 1**: Subtract $100 from Account A.
2. **Step 2**: Add $100 to Account B.

In this case:

* If both steps happen successfully, the transaction is committed (saved).
* If anything fails (e.g., power outage after step 1), the transaction will be rolled back, and it’s like nothing ever happened.

This ensures that no one loses money or data because of incomplete actions.

**Transaction management with SQL**

Transaction management in SQL ensures that all operations within a transaction are completed successfully or, if any error occurs, the operations are rolled back to maintain data consistency.

**Key SQL Statements for Transaction Management:**

1. **BEGIN TRANSACTION** (or START TRANSACTION):
   * Starts a new transaction.
2. **COMMIT**:
   * Confirms the transaction. All changes made during the transaction are saved to the database.
3. **ROLLBACK**:
   * Cancels the transaction. If something goes wrong, all the changes are undone, and the database goes back to the state it was in before the transaction started.
4. **SAVEPOINT**:
   * Creates a point within a transaction where you can roll back to if needed, without undoing the entire transaction.

**Why is Transaction Management Important?**

1. **Data Consistency**: Ensures that data stays accurate and follows rules even when multiple operations are performed.
2. **Error Recovery**: With ROLLBACK, we can easily undo partial changes, preventing database corruption or invalid states.
3. **Concurrency Control**: Transactions isolate operations from each other so that multiple users can safely modify the database simultaneously.

A **transaction log** is a crucial component in databases that records all changes made to the database. It keeps a detailed log of every operation performed during transactions, ensuring that the database can recover from failures (e.g., crashes, power loss) and maintain data integrity.

**Now, what is concurrency control? explained to a kid**

Alright! Imagine you're in a big library with many books, and there are a lot of kids who want to read the same books at the same time. But if everyone just grabs books without following any rules, it could get confusing. Some might read the wrong page, or someone might take the book away while another kid is still reading.

**Concurrency Control in Simple Terms:**

**Concurrency control** is like having a librarian who makes sure everyone takes turns and no one messes up the books while others are reading. It’s about making sure multiple people can use the library (or database) at the same time without causing problems for each other.

In a **database**, the same thing happens. Many people (or programs) might try to read or change the same data at the same time. If there’s no control, it can lead to:

1. **Confusion** (like two kids trying to write in the same book at the same time).
2. **Data getting mixed up** (one kid erases a page another kid just wrote on).
3. **Incorrect results** (one kid adds up numbers while another changes them).

**How Concurrency Control Works:**

Concurrency control makes sure that:

* **Everyone waits their turn**: If one person is changing something, others wait until that change is done.
* **Data stays safe**: Even if multiple people are using the database, no one messes up each other's work.

It uses rules to keep everything in order. Some of these rules are:

1. **Locking**: Like locking a book while one kid is reading it. No one else can change it until that kid is done.
2. **Timestamping**: Like giving each kid a ticket with a number. Whoever has the earlier number gets to go first.
3. **Versioning**: Like making copies of the book for everyone to read, and later combining the changes in an orderly way.

**Example:**

Let’s say two kids, Alice and Bob, are both trying to take $10 from the same piggy bank that has $20. Without concurrency control, they might both see $20, take out $10 each, and now the piggy bank would say $0, even though only $10 was taken out! That's wrong.

With concurrency control, if Alice starts first, Bob has to wait. Once Alice is done, the database updates the piggy bank to $10. Then Bob can take his turn, seeing the correct amount and taking out $10, leaving $0.

**Why is Concurrency Control Important?**

It makes sure that everyone can use the database at the same time without problems, just like kids sharing books in a library without ruining them for each other!

A **deadlock** in a database happens when two or more transactions are stuck, each waiting for the other to release a resource (like a lock), so they can proceed. As a result, none of the transactions can move forward, creating a "standstill."