TRANSACTIONS

AGENDA

- What are transactions?
- Why do we need them?
- ACID
- Atomicity
- Isolation

Rationale behind transactions

- Imagine a scenario where you pay someone from your bank account. Something goes wrong during the payment process and the payment does not go through. HOWEVER, the EFT fee that the bank usually charges is debited from your account.....

- Without DB transactions, the above scenario would be a very real possibility.

What are transactions?

- A set of operations that must succeed or fail as a single unit.
- All operations ALWAYS run in a transactional context.
- By default, transactions contain a single DB operation.
- We can issue commands to the DBMS to begin or end transactions.
- This allows us to group operations into a single atomic unit.

What are transactions?

- Deciding which operations to group into a transaction depends on the context and objectives of the software. In other words, it is a business decision.
- In a typical layered architecture, the transaction boundaries should be demarcated in the business (service) layer.
- Demarcating it in the repository defeats the purpose of a service layer.
- Demarcating it in the controller ties the presentation logic to business logic.

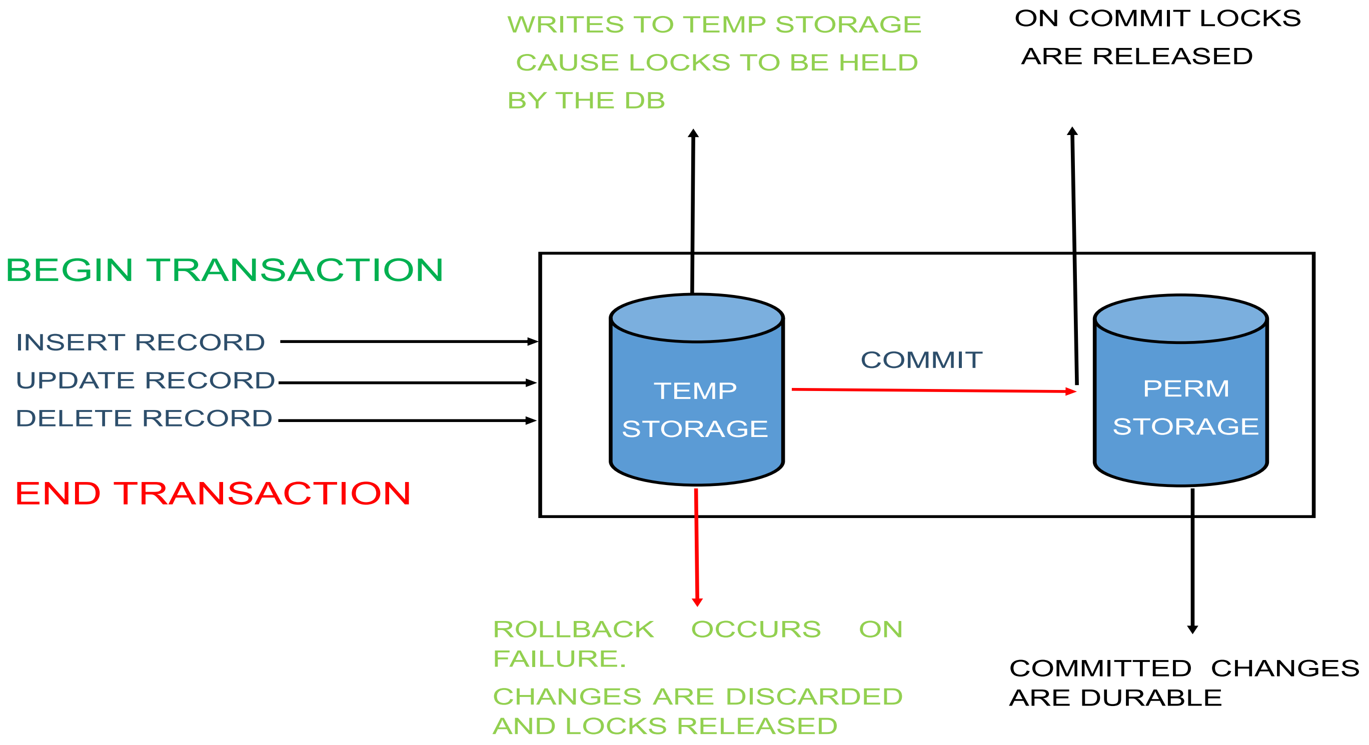
ACID

- The SQL Standard specifies 4 properties that Database transactions must have.
- These are famously known by the acronym ACID.
- Atomicity – All or nothing. This is apparent from the definition of transactions.
- Consistency – Transactions must not violate any of the DB constraints.
- Isolation – Concurrent transactions must not interfere with the current transaction.
- Durability – Completed transactions endure even through failure. (Robustness)
- We will focus on atomicity and isolation.

Atomicity

- To achieve atomicity we need the ability to define a transaction as well as control the rollback of a transaction.
- If all operations succeed all changes materialise and become durable.
- If a single operation fails, all prior changes in the transactional context are undone and the DB is restored to the state it was at before the transaction commenced.
- Every DBMS has its own way of enacting rollback.
- Demos

Atomicity



WRITES TO TEMP STORAGE CAUSE LOCKS TO BE HELD BY THE DB
ON COMMIT LOCKS ARE RELEASED
BEGIN TRANSACTION
INSERT RECORD
UPDATE RECORD
DELETE RECORD
END TRANSACTION
TEMP STORAGE
COMMIT
PERM STORAGE
ROLLBACK OCCURS ON FAILURE.
CHANGES ARE DISCARDED AND LOCKS RELEASED
COMMITTED CHANGES ARE DURABLE

Isolation

- If only one user ever used the DB at a time, there would be no need for this characteristic.
- In the real world, multiple user can try to access and change resources at the same time.
- The pinnacle of isolation is serializable: Outcome as if the transactions were executed sequentially.
- There is tension between isolation strictness and availability.
- The stricter the isolation level, the less performant the system becomes.

Isolation

- Question: Who feels they have a reasonable understanding of isolation?

Isolation

- The SQL standard defines 3 concurrency anomalies or phenomena.
- Dirty Read – reads an uncommitted change
- Non repeatable read – A query is run twice in a transaction and reads different results
- Phantom read – a query yielding multiple results yields more or fewer results.

Isolation

- The Sql standard also defines 4 Isolation levels, based on the MINIMUM anomalies they prevent.
  - Read uncommitted (none)
  - Read committed (dirty reads)
  - Repeatable read (dirty reads and non-repeatable reads)
  - Serializable (All three standard anomalies)
- DBMS providers are free to prevent more anomalies than required for a given level.
- Postgres and MySql prevent phantom reads on repeatable read isolation level.
- Demos

Isolation

- There are 2 main strategies to control isolation:
  - Locking (avoiding conflicts)
  - Version control (detecting/managing conflicts)
- Locking has a higher performance penalty, but is better for data integrity.
- Two types of lock:
  - shared (read) lock, prevents writes, allows reads.
  - exclusive (write) lock, disallowing both read and write operations.
- It is possible to obtain locks explicitly using:
  - Select for share (shared lock)
  - Select for update (exclusive lock)

Isolation

- These isolation levels describe what the current transaction sees about other transactions, not what other transactions see about the current transaction.
- A transaction can always see its own changes.
- There are other anomalies not specified in the Sql standard:
  - Lost update
  - Read skew
  - Write skew
  - Dirty write

Homework

- Write code demonstrating the following using the Spring @Transactional annotation:
  - Dirty read
  - Non repeatable read
  - Phantom read
- Read up more on the other two ACID characteristics: Consistency and Durability.
- Is it possible to achieve a dirty write using any DBMS?
- What is a lost update and how can it be prevented?
- What are read and write skews and how can they be prevented?