Chapter 7 : Transactions

What can go wrong in data systems?

- Database software or hardware may fail at any time.
- Application can crash at any time.
- Interruptions in the network.
- Concurrent clients.
- Data partially updated
- Race conditions can cause bugs.

ACID

- **Atomicity :**

  Describes what happens if a fault occurs after some of the writes have been processed. Then those writes are grouped together into an atomic transaction and the transaction cannot be completed due to a fault, then the transaction is aborted and database must discard or undo any writes it has made so far in the transaction.

- **Consistency :**

  Application specific notion of database being in a good state.

- **Isolation :**

  Concurrently executing transactions are isolated from each other, which means that each transaction can pretend that it is the only transaction running in the entire database.

- **Durability :**

  Promise that once transaction has been committed successfully, any data it has written will not be forgotten, even if there is a hardware fault or the database crashes.

Single-Object and Multi-Object operations

- Atomicity
- Isolation
- BEGIN TRANSACTION and COMMIT statement.

Single-object writes

Network connection is interrupted.

If power fails in middle of overwriting the previous value on disk.

If client sees any partially updated value?

Atomicity : Implemented using a log for crash recovery

Isolation : Using lock on each object

Why do we need multi-object transactions?

- In relational data model, a row in one table has a foreign key reference to a row in another table. It ensures that these references remain valid.
- Document data model lacking joins encourages denormalization. When denormalized information needs to be updated, we need to update several documents in one go.
- In databases with secondary indexes, the indexes need to be updated every time you change a value.

Handling errors and aborts

- If transaction actually succeeded, but network failed.
- If error is due to overload, retrying the transaction will make the problem worse not better.
- It is worth retrying only after a transient error but not a permanent error.

Weak Isolation Levels

- Concurrency issues comes when one transaction reads data that is concurrently modified by another transaction or when two transactions try to simultaneously modify the same data.
- Transaction isolation
- Serializable isolation – database guarantees that transactions have the same effect as if they ran serially. It has a performance cost.
- Non serializable isolation levels

Read Committed

- Two guarantees:
- When reading from the database, you will only see data that has been committed(no dirty reads).
- When writing to the database, you will only overwrite data that has been committed.(no dirty writes).

No dirty reads

- A transaction that reads the data from another transaction which is still in uncommitted state.
- Any writes by a transaction only become visible to others when that transaction commits.
- Why dirty reads are prevented?
  - If transaction needs to update several objects, a dirty read means that another transaction may see some of the updates but not others. Data will be seen in partially updated state which is confusing.
  - If a transaction aborts, any writes that has been made need to be rolled back. If db allows dirty reads, that means a transaction may see data that is later rolled back, which is never actually committed to the database.

No dirty writes

- We don’t know in which order writes happen when two transactions try to concurrently update the same object in a database. If an earlier writes is a part of transaction that has not yet been committed and later writes overwrite an uncommitted value.
- Why dirty write is needed?
  - If transactions update multiple objects, dirty writes can lead to a bad outcome.

Implementing read committed

- Row level locks – holds lock until that transaction is committed or aborted.
- One long running write transaction can force many read-only transactions to wait until the long running transaction has completed.
- Response time is slow and bad for operability.
- Most databases use dirty reads – for every object that is written, the database remembers both the old committed value and new value set by transaction that holds the write lock.
- When transaction is ongoing, any other transactions that read objects are given old value, until the transaction holding the write lock commits the new value.

Summary

- ✓ ACID
- ✓ Single object writes
- ✓ Multi-object writes
- ✓ Weak Isolation levels
- ✓ Read committed Isolation level
- ✓ Dirty reads
- ✓ Dirty writes

Thank You!