10 -Transactions and Concurrency

Tuesday, April 16, 2024 5:22 AM

What is Database Transaction ?

In short, a database transaction is a sequence of multiple operations performed on a database, and all served as a single logical unit of work — taking place wholly or not at all.
In other words, there's never a case that *only half of the operations* are performed and the results saved. When a database transaction is in flight, the database state may be temporarily inconsistent, but when the transaction is committed or ends, the changes are applied.

What are ACID properties, and why are they important?

- **Atomicity**

  Atomicity in terms of a transaction means *all or nothing*. When a transaction is committed, the database either completes the transaction successfully or rolls it back so that the database returns to its original state.

- **Consistency**

  consistency means that your data stays reliable and accurate throughout a transaction. Imagine you're transferring money from one bank account to another. Consistency ensures that during this transfer, your balance doesn't glitch and show two different amounts at the same time. It's like making sure everything adds up correctly so you don't lose track of your money!

- **Isolation**

  With multiple concurrent transactions running at the same time, each transaction should be kept independent without affecting other transactions executing simultaneously. For most database systems, the order of the transactions is not known in advance. Transactions are instead run in parallel, and some form of database locking is utilized to ensure that the result of one transaction does not impact that of another. Typically, databases offer several isolation levels to control the degree of transactional integrity.

- **Durability**

  Durability in ACID means that once a transaction is committed or completed, it stays that way even if there's a power outage or a system crash. It's like saving a document on your computer—once it's saved, you expect it to be there even if your computer suddenly shuts down. Durability ensures that your data remains safe and intact, no matter what happens.

- **NOTE THAT [By Default MYSQL Wrap Every statement in the query and put it into TRANSACTION Statment]**

How to make a Transaction ?

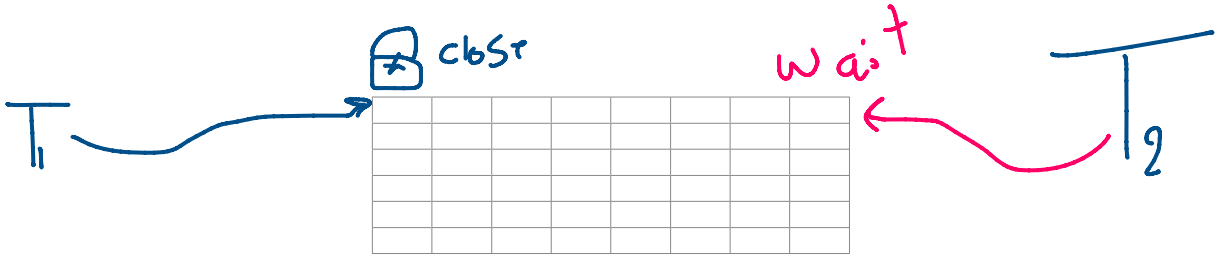
STAER TRANSACTION ;

.....
.....
.....
.....
.....

[1] COMMIT ; -> To Confirm The Transaction
[2] ROLLBACK ; -> To Cancel The Changes and Make Database back to it's previous State

The Default Behave of how MYSQL Handel Concurrency ?

With So simple trick it's lock the row from begin modified while other transaction work in it



Common Concurrency Problems & How To solve it

- **Lost Updates**

  The lost update problem is a concurrency issue that can occur in database systems when multiple transactions attempt to update the same data concurrently. Specifically, it refers to a situation where one transaction's updates to a piece of data are overwritten or lost due to another transaction's conflicting updates.

  Here's a simplified example to illustrate the lost update problem:

  1 - Initial State: Suppose you have a database table with a column named "Balance," initially set to 100.
  2 - Transaction A and Transaction B: Two transactions, let's call them A and B, both read the Balance value (100) simultaneously.
  3 - Updates Made: Transaction A increases the Balance by 50 (resulting in 150), and Transaction B increases it by 30 (resulting in 130).
  4 - Lost Update: Transaction B completes and updates the Balance to 130. However, when Transaction A completes and updates the Balance to 150, it overwrites the change made by Transaction B. As a result, the update from Transaction B is lost, and the final Balance is incorrect (150 instead of 130)

- **Dirty Reads**

  describe a situation where one transaction reads data that has been modified by another transaction but not yet committed. In simpler terms, a dirty read occurs when a transaction reads uncommitted changes made by another transaction.

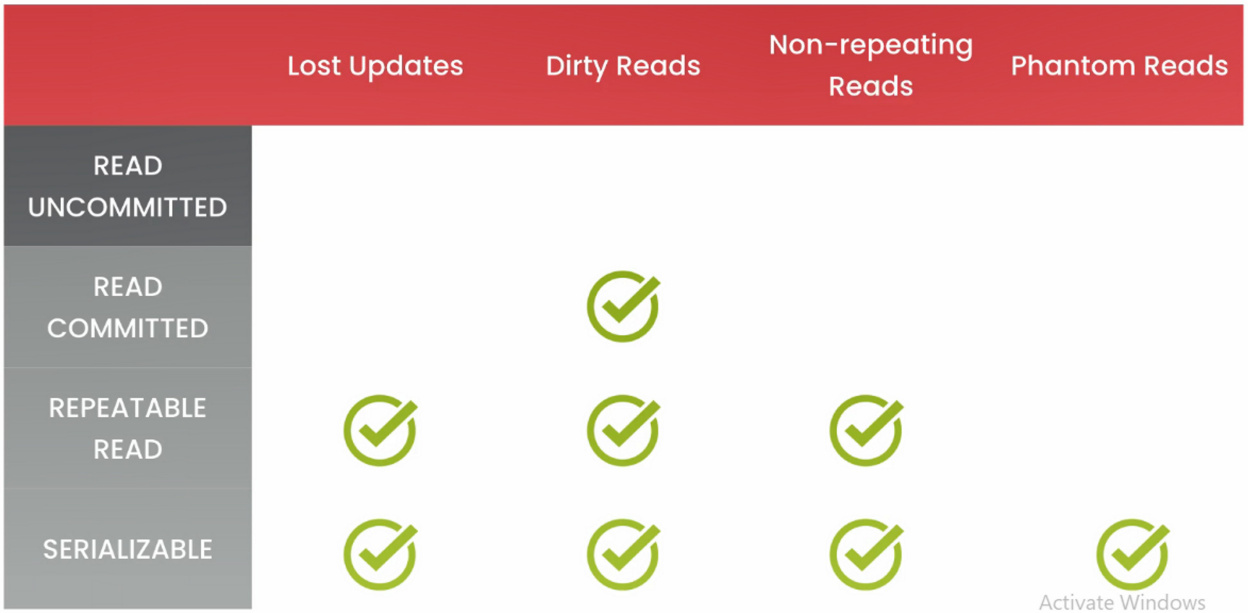
- **Non-Repeating Reads**

  when a transaction reads the same data multiple times but gets different results each time due to concurrent modifications by other transactions. This issue arises from the lack of isolation between transactions and can lead to data inconsistency.

- **Phantom Reads**

  Phantom reads are a phenomenon that can occur in database systems when a transaction retrieves a set of records based on a certain condition, then another concurrent transaction inserts or deletes records that match that condition, causing the first transaction to "see" additional or missing rows when it repeats the same query. This issue arises due to the lack of proper isolation between transactions and can lead to unexpected results and data inconsistency.

Pause



Type Of Isolation Levels That Solve this Problems

- **READ UN COMMITTED**

  Assume You query on table to get values As Transaction T1 and another Transaction T2 update the some records
  T1 Can read the updated values that done using T2 even if T2 ROLLBACK which Mean we read bad and dirty values

```sql
-- Transaction 1
START TRANSACTION ;
UPDATE customers SET points = 10 WHERE customer_id = 1;
ROLLBACK;
```

```sql
-- Transaction 2
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ;

SELECT points FROM customers WHERE customer_id = 1 ;
```

- **READ COMMITTED**

From the name just read the committed values , seems good **but note it may cause problem for reading different values For same variable in 1 Transaction**

```sql
-- Transaction 2
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;

START TRANSACTION ;
SELECT points FROM customers WHERE customer_id = 1 ;
SELECT points FROM customers WHERE customer_id = 1 ;
COMMIT;
```

```sql
-- Transaction 1
START TRANSACTION ;
UPDATE customers SET points = 20 WHERE customer_id = 1;
COMMIT;
```

→ default in MY SQL

- **REPEATABLE READ**

Solve the above problem the values will be consistent , but will bring another problem
Say you make query to do an action based on a condition will this isolation level
Assume T1 make an update but not committed it yet on ID = X
And T2 query based on condition and before T1 X not valid but after work of T1 ID = X become valid
But the problem is that is Record with ID = X not include in the second T2 because it's not committed yet

And that's what called with **Phantom Reads** to solve it you must switch to another isolation level called **SERIALIZABLE**

- **SERIALIZABLE**

It's the highest level of isolation and actually no concurrency here as it work like that
The Transaction have a awareness about all other transactions that may affect the result of it's query so it will wait until all of them are commit or rollback and then it's continue it's execution

```sql
-- Transaction 1
START TRANSACTION ;
UPDATE customers SET state = 'VA' WHERE customer_id = 1;
COMMIT;
```

→ Update record affect the Result

→ wait until it's done

customers

Limit to 50 rows

```sql
-- Transaction 2
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;

START TRANSACTION ;
SELECT * FROM customers WHERE state = 'VA';
COMMIT;
```

How To Change Isolation Level in SQL ?

SHOW VARIABLES LIKE 'transaction_isolation'

SET [optional](SESSION - GLOABLE) TRANSACTION ISOLATION LEVEL SERIALIZABLE ;

[NOTE]

THE DEFAULT IS APPLYING TO THE NEXT TRANSACTION