ISOLATION in DB

agenda: To understand the concept of isolation in dbs to solve the problem of concurrency.

ACID

Isolation

Q: What is isolation ?

---> Isolation says that, if there are 2 or more than 2 txns running at the same time, each one should behave as if it ran alone.

If a db is ACID compliant then it has built in capabilities to control isolation.

Q: What can happen if we don't take care of the isolation levels between multiple transactions ?

Accounts

id	name	balance
1	Sanket	1000
2	Sarthak	1000

Assume initial state of a table 'Accounts'

Txn 1

Start transaction;

Accounts

id	name	balance
1	Sanket	500
2	Sarthak	1000

UPDATE accounts SET
balance = balance - 500
WHERE id = 1;

// some issue

Accounts

id	name	balance
1	Sanket	1000
2	Sarthak	1000

Rollback;

Txn 2

Start transaction;

SELECT balance FROM accounts
WHERE id=1;

500(result of the query)

It is trying to read an uncommited value/data

Because txn1 rolledback
we have read an invalid value

This problem is called as Dirty Read problem

Txn 1

Start transaction;

Accounts

id	name	balance
1	Sanket	1000
2	Sarthak	1000

SELECT balance FROM
accounts WHERE id=1;
// result -> 1000

// more operations going on
// but non changed value of balance

Accounts

id	name	balance
1	Sanket	900
2	Sarthak	1000

SELECT balance FROM
accounts WHERE id=1;
// result -> 900

In the same transaction, we are reading a data twice,
and getting diff values.

txn 2

Start transaction;

UPDATE accounts SET
balance=balance-100
WHERE id=1;

Commit;

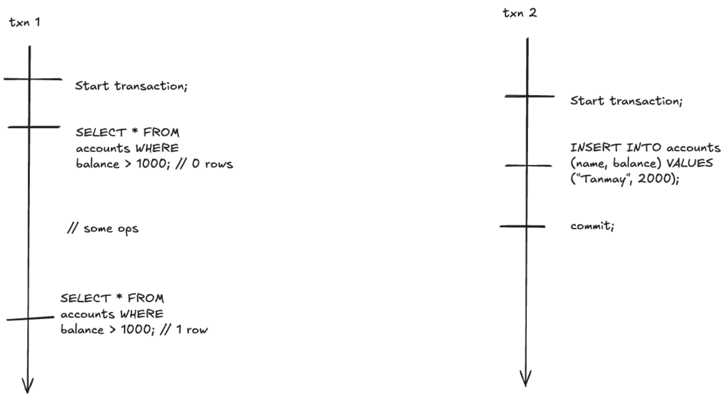
Accounts

id	name	balance
1	Sanket	900
2	Sarthak	1000

This problem is called as "Non repeatable read problem", a txn reads same row
twice but gets diff values.

Accounts

id	name	balance
1	Sanket	1000
2	Sarthak	1000



txn 1

Start transaction;

SELECT * FROM
accounts WHERE
balance > 1000; // 0 rows

// some ops

SELECT * FROM
accounts WHERE
balance > 1000; // 1 row

txn 2

Start transaction;

INSERT INTO accounts
(name, balance) VALUES
("Tanmay", 2000);

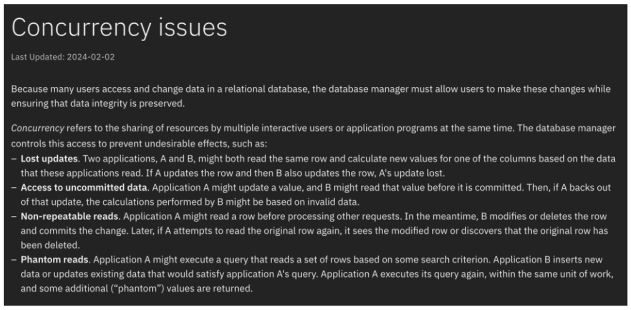
commit;

Accounts

id	name	balance
1	Sanket	1000
2	Sarthak	1000
3	Tanmay	2000

Here the number of rows have changed for the same select query in the same txn.

This problem is called as phantom read problem.



Concurrency issues

Last Updated: 2024-02-02

Because many users access and change data in a relational database, the database manager must allow users to make these changes while ensuring that data integrity is preserved.

Concurrency refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- **Lost updates**. Two applications, A and B, might both read the same row and calculate new values for one of the columns based on the data that these applications read. If A updates the row and then B also updates the row, A's update lost.
- **Access to uncommitted data**. Application A might update a value, and B might read that value before it is committed. Then, if A backs out of that update, the calculations performed by B might be based on invalid data.
- **Non-repeatable reads**. Application A might read a row before processing other requests. In the meantime, B modifies or deletes the row and commits the change. Later, if A attempts to read the original row again, it sees the modified row or discovers that the original row has been deleted.
- **Phantom reads**. Application A might execute a query that reads a set of rows based on some search criterion. Application B inserts new data or updates existing data that would satisfy application A's query. Application A executes its query again, within the same unit of work, and some additional ("phantom") values are returned.

These issues can be solved by controlling the isolation levels.

Solutions provided by MySQL:

Isolation levels.

What is meant by isolation levels ? Isolation level means that what degree of independence we will give to two txns.

MySQL gives 4 level of isolations in general:

READ UNCOMMITTED (Lowest possible isolation)

This is like saying there is no isolation at all. All the above problems like phantom reads, dirty reads, non repeatable reads etc anything can happen.

This is as bad as saying there is no isolation at all.

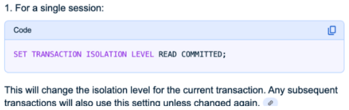
For the further isolation levels mysql has to make extra efforts to make that happen. If there is a situation where only reads are gonna happen and no writes/updates/delete then this is a good option to go for as it will keep things fast.
May be for analytics purpose.

READ COMMITTED

1. In read committed isolation level dirty reads will not happen. Rest other problem still exist.

2. Two identical reads in 1 txn may still give you diff values (non repeatable reads problem) but two txns will never see an uncommitted data from other.

--- SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

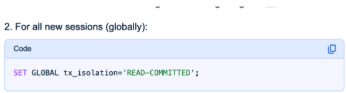


1. For a single session:

Code

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

This will change the isolation level for the current transaction. Any subsequent transactions will also use this setting unless changed again.



2. For all new sessions (globally):

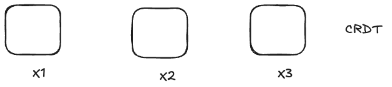
```
SET GLOBAL tx_isolation='READ-COMMITTED';
```

REPEATABLE READS

In this isolation level neither the dirty read nor the non-repeatable read problem will occur. Phantom reads can occur.

SERIALIZABLE (Highest isolation)

In serializable all the above problems will be solved, and it will behave as if two txn are executed one after another instead of parallel.



1- Pessimistic locking

- Isolation level - Read commited LOCK on the row

SELECT ..... FOR UPDATE

txn1 -> A(lock) and B

txn2 -> B(lock) and A

Deadlock

txn1

Start transaction;

Update accounts set balance = balance - 100 where id = 1; // reduce from sanket lock

Update accounts set balance = balance + 100 where id = 2; // add to sarthak

Commit;

txn2

Start transaction;

Update accounts set balance = balance - 200 where id = 2; // reduce from sarthak lock

Update accounts set balance = balance + 200 where id = 1; // add to sanket

Commit;

2 - optimistic locking

keep a version number

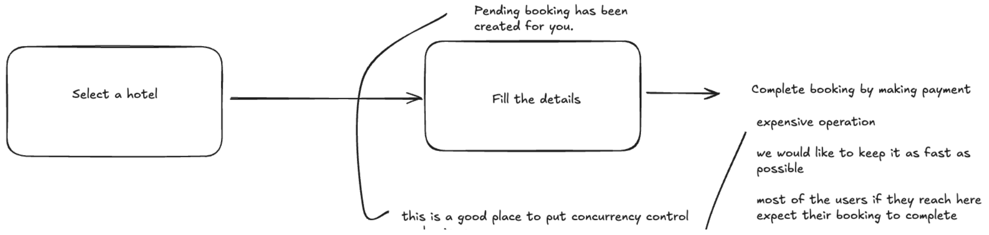
In our table, we will keep a version number and we are not going to take any lock. if a row is updated we update its version number.

Here we allow multiple txns to happen parallely, and isolation: read committed.

If any txn goes through then before we commit, we update the version number to +1.

While other parallel txn before commiting are going to check if the version number they have is same as the version number at the current point of time in db, if yes then commit if not, then rollback.

For hotel room count, we can check the remaining count of the rooms



Select a hotel

Pending booking has been created for you.

Fill the details

Complete booking by making payment

expensive operation

we would like to keep it as fast as possible

most of the users if they reach here expect their booking to complete

this is a good place to put concurrency control mechanisms

we dont want any expensive ops here
hence any serilizable txn
or any locking cannot be afforded here

we dont want bad user experience here
so we cant do optimisitc locking
also.

distributed cache based lock

Lock on redis cache.

app logic
service logic

user airbnb

1. 10

TTL (time to live)