12. Transaction Processing

Most of content here is borrowed from following sources-
>> Lecture slides of Jennifer Widom, Stanford (coursera)
>> Book Elmasri/Navathe

SQL commands related to Transactions

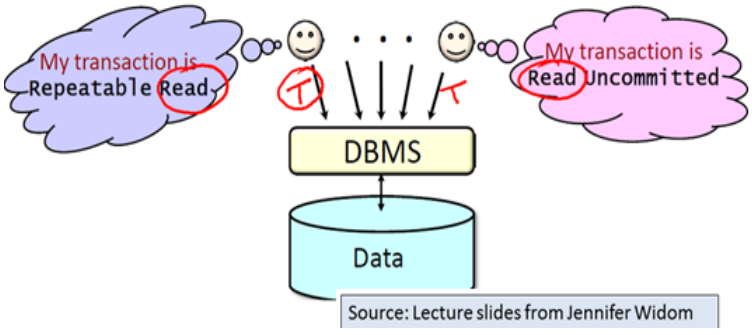
BEGIN TRANSACTION
COMMIT TRANSACTION
ROLLBACK TRANSACTION
SET TRANSACTION
- ISOLATION LEVEL
- ACCESS MODE

Syntax of specifying Isolation levels -

Set Transaction Isolation Level *<isolation level>*;

SQL Isolation Levels

- While serializability is desirable; it has following side effects -
  - Brings in additional overhead (processes that ensure serializable execution schedules)
  - Provides reduced concurrency; serialization algorithms may ask some other transaction to wait, and even aborts in some cases.
- However Serializability may not be required all the time, particularly for reads. Therefore RDBMS implementation also "weaker isolations".
  - Idea of weaker isolation level is – transactions that are interested only in reads, can carry on their work with older values, or even with dirty reads!
- Following are four transaction isolation levels in SQL-92-
  - Read Uncommitted
  - Read Committed
  - Repeatable Reads
  - Serializable [we already know this]
- They are in the order of increased isolation.
- Increased isolation also means increased computation overhead. Higher the isolation lesser will be concurrency, that is
  - delayed scheduling of transaction operation, or even
  - some are getting denial of execution
- Isolation-levels are specified for a transaction
  - Different transaction may execute at different isolation levels
  - Isolation variations are always in terms of READs; that is you may read something that is not as per serializability principles
  - As per SQL standard, Serializable is the default isolation level. However a RDBMS may have different level as default.
  - For example PostgreSQL has read committed as default isolation level.



Problems with lesser Isolation Levels

Read Uncommitted

- OK reading uncommitted data
- Dirty reads may occur
  Example:

```
T1:   UPDATE emp SET salary = 1.1*salary;
      COMMIT   //ROLLBACK;
T2:   SELECT average(salary) FROM emp;
```

- If T2 runs at "Read UnCommitted" level, then it allows reading data from T1 even before it commits. If T1 decides to abort then report of T2 is not as per "serialization principles" - "DIRTY READ" is occurring.
- Even when T1 commits, T2 might read some old data before T1 updates and some updated that is also not yielding "Serializable" execution.

Read Committed

- Reads only committed data at the time of "reading" it.
- However "inconsistent read" might occur. For example a transaction reads a data item. Some other transaction writes X and commits. Now if reads X again then it sees different value of X.
- Note that such a transaction is not serializable. Check and see if it forms a cycle in precedence graph?

Inconsistent reads are also referred as "Non-Repeatable reads". Inconsistent reads may have problems in some cases. Observe behavior of following execution schedules.

Example-1

```
T1: begin transaction;
T2: begin transaction;
T2: select salary from employee where ssn = '101' ;  //50000
T1: update employee set salary = 45000 where ssn = '101';
T2: select salary from employee where ssn = '101' ;  //50000
T1: commit;
T2: select salary from employee where ssn = '101' ;  //45000
T2: commit;
```

Example-2:

```
T1:   UPDATE emp SET salary = 1.1*salary;
T2:   SELECT average(salary) FROM emp;
T1:   COMMIT
T2:   SELECT max(salary) FROM emp;
```

- Problem: T2 gets different values for average? Computes average with different database states in two consecutive requests?
- Reason: NOT SERIALIZABLE; conflicting operations are in different order?

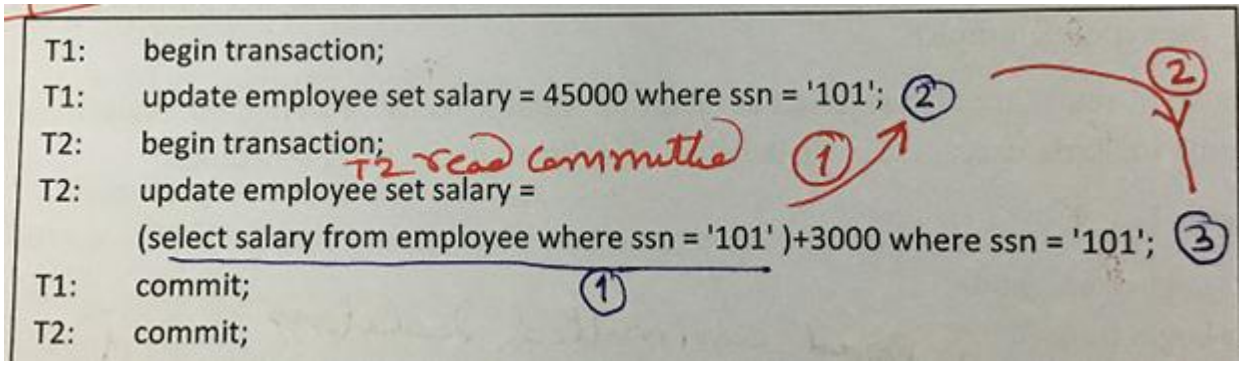
When we are allow this transaction T reading all committed data by simultaneous transactions! We get inconsistent values of a data item in a transaction

Example-3

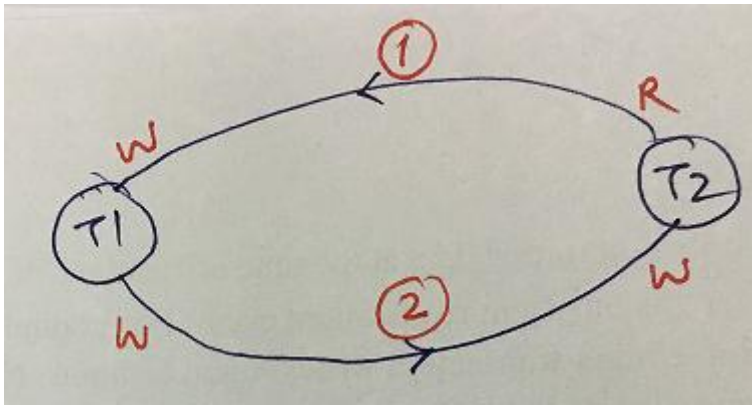
```
T1:   begin transaction;
T1:   update employee set salary = 45000 where ssn = '101';
T2:   begin transaction;
T2:   set transaction isolation level read committed;
T2:   update employee set salary = salary +3000 where ssn = '101';
T1:   commit;
T2:   commit;
```

Can you estimate final update of salary? It produces results as expected 48000!

Example-4 (Some tricky example)



Final update to salary is 53000, which is incorrect!



Due to lower isolation level cycle is allowed to be executed and results incorrect update.

You can simulate this in postgresql

1. Open two sql windows; let us say w1 and w2
2. Type all commands of T1 in w1; and all commands of T2 in w2
3. Issue the commands in above sequence in respective windows!

Again cause of problem is NON-Repeatable READ.

Repeatable Reads Isolation level

- Repeated read of a "data item" is consistent- of course we read only committed; and this level guarantees consistent reads.
- This repeatable read should be applicable to a granularity level of the context
- However repeatable read does not give repeatable read of a relation with respect to new inserted tuples! Consider example below

In example-4, if Isolation level of T2 is set to be repeatable read, DBMS will not let T2 to commit, and thus way will be able to ensure above concurrency problem! T2 will be "aborted" by concurrency control unit of DBMS.

Example-5:

```
T1:   begin transaction;
T1:   INSERT INTO emp VALUES( … ); say adds 10 rwos
T2:   begin transaction;
T2:   set transaction isolation level repeatable read;
T2:   SELECT average(salary) FROM emp;
T1:   COMMIT
T2:   SELECT max(salary) FROM emp;
```

Isolation Level Summary

Isolation Level/Problem	Dirty Read	Non-repeatable reads	Phantom rows
Read Uncommitted	Y	Y	Y
Read Committed	N	Y	Y
Repeatable Read	N	N	Y
Serializable	N	N	N

- Standard default: Serializable
- Weaker isolation levels
  - Increased concurrency + decreased overhead = increased performance
  - Weaker consistency guarantees
  - Default isolation levels Some systems have default Repeatable Read
- Isolation level are for a transaction
  - Each transaction's reads must conform to its isolation level