Transactions

Isolation Level

<http://dev.mysql.com/doc/refman/5.6/en/innodb-concepts.html>

(Section 14.5)

http://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html

(Section 5.1.4 System Variables)

Look at tx\_isolation variable that has session and global scope

Show **global** variables like ‘%tx%’;

Show **session** variables like ‘%tx%’;

Show variables like ‘%tx%’;

Set **GLOBAL | SESSION** transaction isolation level **xxx**;

**Scope of Transaction Characteristics**

You can set transaction characteristics

1. Globally: All subsequent sessions.
2. Session : For the current session

<http://dev.mysql.com/doc/refman/5.6/en/sql-syntax-transactions.html>

(Section 13.3)

START TRANSACTION | BEGIN statement begins a new transaction, disables autocommit (= 0)

COMMIT commits the current transaction, making its changes permanent.

ROLLBACK rolls back the current transaction, undoing any changes.

Undo log in MySQL (physically located within ibdata1 file) is used to keep track of the changes that current transactions is making.

If Tx :

* **Commits**: undo log entry deleted as it is no longer required
* **Rollbacks**: undo log used to get old/previous values of modified data

START TRANSACTION;

Use db3;

select empno, ename, salary from emp;

update emp set salary=salary\*1.2 where deptno =20;

select empno, ename, salary from emp;

ROLLBACK **|** COMMIT;

Some statements cannot be rolled back. Such as create or drop databases, create, drop, or alter tables or stored routines.

<http://dev.mysql.com/doc/refman/5.6/en/innodb-transaction-model.html>

(Section 14.2.8) 5.6

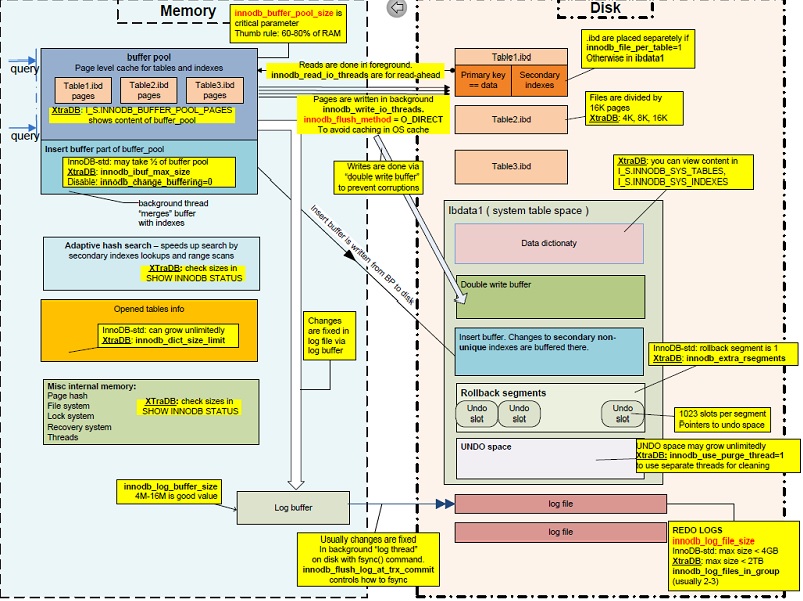
SQL Statements and effect on locks

* Select…. -> Sets no locks, consistent read
* Insert, Delete & Update -> Sets write lock

New commands are

* Select….LOCK IN SHARE MODE ->sets read lock
* Select….FOR UPDATE -> sets write lock

Locks are **released** when the transaction **commits** or is **rolled back**.



**Repeatable Read** Isolation level:

Tx is given a snapshot (version/copy) of database as it exists at the time Tx starts - consistent reads

Transaction **READS** are consistent but not necessarily [Inserts / Updates / Deletes s](http://dev.mysql.com/doc/refman/5.5/en/glossary.html#glos_dml)tatements

A Select doesn’t set any lock, so other tx are free to modify same data while a read is being performed

Open Mysql console window

Show **session** variables like ‘%tx%’; **#** Repeatable Read

**show engine innodb status;**

**Create table innodb\_lock\_monitor(a int) engine=innodb;**

**set global innodb\_status\_output=0;**

**show full processlist;**

**Drop Database Concurrency; Create Database Concurrency; Use Concurrency;**

Drop table t;

Create Table T (Field1 integer, Field2 integer) Engine=InnoDB;

**Notice : No primary key**

|  |  |  |
| --- | --- | --- |
| Time | T1 | T22 |
| 1 | **Start Tx;** |  |
| 2 | Snapshot | **Start Tx;** |
| 3 | Select \* from T; | Snapshot |
| 4 | {empty set} |  |
| 5 |  | Select \* from T; |
| 6 |  | {empty set} |
| 7 | Check logs TXid,  no locks on Select  No undo entry | Check logs TXid, no locks on Select  No undo entry |
| 8 |  | Insert into T values (1, 2); |
| 9 |  | Check logs TXid ,Undo Entry  1 Lock Struct, 0 row lock(s)  TABLE LOCK lock mode IX |
| 10 | Select \* from T; |  |
| 11 | {empty set} |  |
| 12 |  | Select \* from T; |
| 13 |  | {1,2} |
| 14 |  | **Commit**; *Release locks* |
| 15 | Select \* from T; |  |
| 16 | {empty set} |  |
| 17 | Insert into T values (1, 2); |  |
|  | Check logs TXid ,Undo Entry  1 Lock Struct, 0 row lock(s)  TABLE LOCK lock mode IX |  |
| 18 | {1,2} |  |
| 19 | **Commit**; *Release Locks* |  |
| 20 | Select \* from T; |  |
| 21 | 2 rows {(1,2),(1,2)} |  |

Drop table t;

Create Table T (Field1 integer, Field2 integer, primary key(field1) ) Engine=InnoDB;

|  |  |  |
| --- | --- | --- |
| Engine=InnoDB;Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | SnapShot | **Start Tx;** |
| 3 | Select \* from T; | Snapshot |
| 4 | {empty set} |  |
| 5 |  | Select \* from T; |
| 6 |  | {empty set} |
| 7 | Check logs TXid  no locks on Select  No undo entry | Check logs TXid  no locks on Select  No undo entry |
| 8 |  | Insert into T values (1, 2); |
| 9 |  | Check logs TXid ,Undo Entry  1 Lock Struct, 0 row lock(s)  TABLE LOCK lock mode IX |
| 10 | Select \* from T; |  |
| 11 | {empty set} |  |
| 12 |  | Select \* from T; |
| 13 |  | {1,2} |
| 14 | Select \* from T; |  |
| 15 | {empty set} |  |
| 16 | Insert into T values (1, 2);  **WAITING** |  |
| 17 |  | Commit; | Rollback; |
| 18 | **If T2 Commit:**  ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY' |  |
|  | **If T2 Rollback:**  Query OK, 1 row affected |  |

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key(field1) ) Engine=InnoDB;

Insert into t values (2,2);

Set session transaction isolation level Repeatable Read;

Show session variables like ‘%tx%’;

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | SnapShot | **Start Tx;** |
| 3 | Select \* from T; | Snapshot |
| 4 | {2,2} |  |
|  |  | Select \* from T; |
|  |  | {2,2} |
|  | Check logs : TXid | Check logs : TXid |
| 5 | Update t set field2 = 3  where field1 = 2; |  |
|  | Check logs TXid , 1 Undo Entry  2 Lock Struct, 1 row lock(s)  TABLE LOCK lock mode IX |  |
| 6 |  | update t set field2 = 4  where field1 = 2; |
| 7 |  | **Waiting**.. Check logs :  TXid, Ex lock, rows locked, Undo log, waiting |

Possible Outcomes:

1. **If T1 commits**, it writes (2,3) to db and releases Ex lock, Then T2 (Who was waiting) gets Ex lock and writes (2,4) to the database; - > Lost update

2. **If T1 rollbacks**, it releases Ex lock, Then T2 (Who was waiting) gets Ex lock and writes (2,4) to the database;

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key(field1) ) Engine=InnoDB;

Set session transaction isolation level Repeatable Read;

Show session variables like ‘%tx%’;

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | Snapshot | **Start Tx;** |
| 3 | Insert into t values (2,2),(3,3),(4,4),(5,5); | Snapshot |
| 4 | Select \* from T; {2,2}{3,3}{4,4}{5,5} |  |
| 5 |  | Select \* from T;  Empty set |
|  | Check logs : TXid  Undo Entry | Check logs: TXid |
| 6 | **Commit**; *Release Locks* |  |
|  |  | **Delete** from T where field1=2;  **{1 Row Affected}** |
| 7 |  | Select \* from T;  {Empty set}– No Phantom Read for Deletes |
| 8 |  | **Commit**; |
| 9 |  | Select \* from T; {3,3}{4,4}{5,5} |

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key(field1) ) Engine=InnoDB;

Set session transaction isolation level Repeatable Read;

Show session variables like ‘%tx%’;

**Phantom Read on Update of one row**

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | Snapshot | **Start Tx;** |
| 3 | Insert into t values (2,2),(3,3),(4,4),(5,5); | Snapshot |
| 4 | Select \* from T; {2,2}{3,3}{4,4}{5,5} |  |
| 5 |  | Select \* from T;  {Empty set} |
|  | Check logs : TXid  4 Undo Log Entries,  0 Row Locks  1 Lock Struct – IX  Intention Lock | Check logs: TXid |
| 6 | **Commit**; |  |
| 7 |  | **Update** t set field2=10 where field1=2; |
| 8 |  | Select \* from T;  {2,10} - Phantom Read **JUST for updated rows** |
| 9 |  | **Commit**; |
| 10 |  | Select \* from T; {2,10}{3,3}{4,4}{5,5} |

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key(field1) ) Engine=InnoDB;

Set session transaction isolation level Repeatable Read;

Show session variables like ‘%tx%’;

**Phantom Read on Update of all affected rows**

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | Snapshot | **Start Tx;** |
| 3 | Insert into t values (2,2),(3,3),(4,4),(5,5); | Snapshot |
| 4 | Select \* from T; {2,2}{3,3}{4,4}{5,5} |  |
| 5 |  | Select \* from T;  {Empty set} |
|  | Check logs : TXid  4 Undo Log Entries,  0 Row Locks  1 Lock Struct – IX  Intention Lock | Check logs: TXid |
| 6 | **Commit**; |  |
| 7 |  | **Update** t set field2=10; |
| 8 |  | Select \* from T;  {(2,10),(3,10),(4,10),(5,10} - Phantom Read **for ALL updated rows** |
| 9 |  | **Commit**; |

Repeatable Read

Phantom Read

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key(field1) ) Engine=InnoDB;

Insert into t values (1,1),(2,2),(3,3);

Set session transaction isolation level Repeatable Read;

Show session variables like ‘%tx%’;

|  |  |  |
| --- | --- | --- |
| Time | Tx A | Tx B |
| 1 | Start Tx; |  |
| 2 | SnapShot | Start Tx; |
| 3 | Select \* from table;  (1,1),(2,2),(3,3) | SnapShot |
| 4 |  | select \* from Table;  (1,1),(2,2),(3,3) |
| 5 |  | Insert into table values (4,4); |
| 6 | Select \* from table;  (1,1),(2,2),(3,3) |  |
| 7 |  | Commit; |
| 8 | Select \* from table;  (1,1),(2,2),(3,3) |  |
| 9 | **Update** t Set field2=5;  **Four rows changed;** |  |
| 10 | Select \* from table;  (1,**5**),(2,**5**),(3,**5**), **(**4**,5)**  **Phantom Read for updated rows** |  |
| 11 | Commit; |  |

**Committed Read** isolation level:

Drop table t; Create Table t ( Field1 integer, Field2 integer, Primary key (field1) ) Engine=InnoDB;

Set session transaction isolation level Read Committed;

Show session variables like ‘%tx%’;

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | Start Tx; |  |
| 2 |  | Start Tx; |
| 3 | Select \* from T; |  |
| 4 | {empty set} |  |
| 5 |  | Select \* from T; |
| 6 |  | {empty set} |
| 7 |  |  |
| 8 |  | Insert into T values (1, 2); |
| 9 | Select \* from T; |  |
| 10 | {empty set} |  |
| 11 |  | Select \* from T; |
| 12 |  | {1,2} |
| 13 |  | Commit; |
| 14 | Select \* from T; |  |
| 15 | {1,2} |  |
| 16 | Update T Set Field2=10  Where Field1=1; |  |
| 17 |  | Select \* from T; {1,2}  cannot see uncommitted data |
| 18 | Commit; |  |
| 19 | Select \* from T; | Select \* from T; {1,10} |

**Uncommitted Read** isolation level:

Drop table t; Create table t (Field1 integer, Field2 integer, primary key (field1) ) Engine=InnoDB;

Set session transaction isolation level Read Uncommitted;

Show session variables like ‘%tx%’;

Allows user see uncommitted data but cannot do anything with new values until uncommitted tx commits | Rollbacks

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 |  | **Start Tx;** |
| 3 | Select \* from T; |  |
| 4 | {empty set} |  |
| 5 |  | Select \* from T; |
| 6 |  | {empty set} |
| 7 |  |  |
| 8 |  | Insert into T values (1, 2); |
| 9 |  |  |
| 10 | **Select \* from T;** |  |
| 11 | **{1,2}** |  |
| 12 | Update T Set Field2=10  Where Field1=1;  Waiting to see what happens to T2 | Select \* from T;  {1,2} |
| 13 |  | **Commit**; or **Rollback** |
| 14 |  |  |
| 15 | Select \* from T; |  |
| 16 | **{1,10}** or **{empty set}** |  |
| 19 | **Commit**; |  |

Serializable isolation level:

Drop table t; Create table t (Field1 integer, Field2 integer, primary key (field1) ) Engine=InnoDB;

Show session variables like ‘%tx%’;

Set session transaction isolation level Serializable;

SELECT ... statements get converted to

SELECT ....WITH SHARE LOCK -> Read Lock

INSERT, DELETE & UPDATE -> Write lock on the row.

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 |  | **Start Tx;** |
| 3 | Select \* from T; |  |
| 4 | {empty set} Read Lock |  |
| 5 |  | Select \* from T; |
| 6 |  | {empty set} Read Lock |
| 7 |  |  |
| 8 |  | Insert into T values (1, 2);  Waiting….Timeout - Wr Lock |
| 9 |  |  |
| 10 | Select \* from T; |  |
| 11 | {empty set} |  |
| 12 | Insert into T values (2, 1);  Waiting on Wr Lock  Deadlock – Tx Rollback by DB  Show Enginne innodb status |  |
| 13 |  | **Commit**; |

MySQL killed transaction that looked for write lock most recently, newer Tx. by killing Tx, it release the Read lock

Repeatable Read

Use Concurrency;

Drop Table Flights;

Create Table Flights (

Seat Integer,

Occupied Integer,

Passenger\_Name varchar(20),

Primary Key (Seat)

) Engine = innodb;

insert into Flights values(22,0,null);

insert into Flights values(23,1 ,’George Clooney’);

insert into Flights values(24,0,null);

insert into Flights values(25,0,null);

Set session transaction isolation level Repeatable Read; Show session variables like ‘%tx%’;

|  |  |  |
| --- | --- | --- |
|  | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | Snapshot | **Start Tx;** |
| 3 | select \* from Flights where  occupied = 0;(repeatable read, no locks->ok) | Snapshot |
| 4 |  | select \* from Flights where  occupied = 0;(repeatable read, no locks->ok) |
| 5 | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘Mary Dooley’ where seat = 22; -ok  (exclusive lock row) |  |
| 6 |  | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘John Jones’ where seat = 22; -(try to lock same row, wait,)  ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction |

But it transpires that the last user to commit ( John) has actually got the seat!!! John overwrites Mary’s name !!! Scenario same as page 7 above

Lost update problem!!!

We can overcome Lost Updates using

Select ......For Update ->Write lock

|  |  |  |
| --- | --- | --- |
|  | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 | SnapShot | **Start Tx;** |
| 3 | select \* from Flights where  occupied = 0 **for update**; (repeatable read,  write locks->ok) | Snapshot |
| 4 |  | select \* from Flights where  occupied = 0 **for update** ; (repeatable read,  write locks->waiting) ERROR 1205 (HY000): Lock wait timeout  exceeded; |
| 5 | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘Mary Dooley’ where seat = 22; -ok  (exclusive lock row) |  |
| 6 | Commit; |  |
| 7 |  | select \* from Flights where  occupied = 0 **for update**; (repeatable read, write locks-  >ok) |

Serializable

Use concurrency;

Drop table flights;

create table Flights(

seat integer,

occupied integer,

passenger\_name varchar(20), Primary Key (Seat)

) Engine = innodb;

insert into Flights values(22,0,null);

insert into Flights values(23,1,’George Clooney’);

insert into Flights values(24,0,null);

insert into Flights values(25,0,null);

set session transaction isolation level serializable;

|  |  |  |
| --- | --- | --- |
|  | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 |  | **Start Tx;** |
| 3 | select \* from Flights where  occupied = 0 **;**  (Converted to read lock ->ok) |  |
| 5 |  | select \* from Flights where  occupied = 0 ;  (Converted to read lock ->ok) |
| 6 | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘Mary Dooley’ where seat = 22; -  wait  (looking for exclusive lock row) |  |
| 7 |  | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘John Jones’ where seat = 22; -wait  (looking for exclusive lock) ERROR Deadlock Kill Tx |
|  | Query ok,  1 row affected |  |

No lost update but Deadlock as T2 gets terminated

MySQL killed transaction that looked for write lock most recently, newer Tx. by killing Tx, it release the Read lock

|  |  |  |
| --- | --- | --- |
|  | T1 | T2 |
| 1 | **Start Tx;** |  |
| 2 |  | **Start Tx;** |
| 3 | select \* from Flights where  occupied = 0 **for update;**  (converted to write lock ->ok) |  |
| 4 |  | select \* from Flights where  occupied = 0 **for update**; (waiting...) |
| 5 | seat 22 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘Mary Dooley’ where seat = 22; -  (exclusive lock row) |  |
| 6 | Commit; |  |
| 7 |  | select \* from Flights where  occupied = 0 **for update**; |
|  |  | seat 24 is available, grab it:  update Flights set occupied = 1, passenger\_name = ‘John Jones’ where seat = 24; -ok |

Table Locks <http://dev.mysql.com/doc/refman/5.5/en/lock-tables.html>

(Section 13.3.5)

MySQL uses

 MyISAM, memory tables: Table-level locking

 InnoDB : Row-level locking for tables.

InnoDB uses **automatic** row-level locking

InnoDB storage engine handles most locking issues without user involvement

(Section 8.10.2)

Lock Tables explicitly acquires table locks for the current client session.

You must have the [LOCK TABLES priv](http://dev.mysql.com/doc/refman/5.5/en/privileges-provided.html#priv_lock-tables)ilege, and the

[SELECT p](http://dev.mysql.com/doc/refman/5.5/en/privileges-provided.html#priv_select)rivilege for each object to be locked.

Locks1.sql

use concurrency;

START TRANSACTION;

LOCK TABLES flights READ;

Select \* from flights; #no-one can write to this table

UNLOCK TABLES;

COMMIT;

|  |  |  |
| --- | --- | --- |
|  | T1 | T2 |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

Locks2.sql

START TRANSACTION;

LOCK TABLES flights WRITE;

Update flights set occupied = 1 where seat =24;

UNLOCK TABLES;

COMMIT;

<https://www.adayinthelifeof.nl/2010/12/20/innodb-isolation-levels/>

show engine innodb status;

show full processlist;

<http://dev.mysql.com/doc/refman/5.0/en/innodb-locks-set.html>

<http://dba.stackexchange.com/questions/15854/innodb-row-locking-how-to-implement>

<http://www.w3resource.com/mysql/mysql-show.php>

<http://courses.cs.washington.edu/courses/csep544/14wi/calendar/lecturelist.html>

<http://courses.cs.washington.edu/courses/cse344/12wi/sections/transactions.txt>

<http://courses.cs.washington.edu/courses/cse344/12au/lectures/lecture18-transactions.txt>

<http://stackoverflow.com/questions/4034976/difference-between-read-commit-and-repeatable-read>