MySQL Isolation Level

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Problems prevented</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>None</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>Dirty reads</td>
</tr>
<tr>
<td>REPEATABLE READ</td>
<td>Dirty reads, lost updates, nonrepeatable reads</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>All</td>
</tr>
</tbody>
</table>

Example

Set the level to SERIALIZABLE for the next transaction
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Set the level to READ UNCOMMITTED for the current session
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Set the level to READ COMMITTED for all sessions
SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED

- Dirty Read or Uncommitted Read

The problem when one SQL users see data that another user has not yet committed is called a dirty read or uncommitted read.

Example

1. User U1 wants to increase the amount of penalty.

   UPDATE PENALTIES
   SET AMOUNT = AMOUNT + 25
   WHERE PAYMENTNO = 4;

2. Before U1 ends the transaction with a COMMIT statement, user U2 accesses the same penalty.

   SELECT *
   FROM PENALTIES
   WHERE PAYMENTNO = 4;

3. U1 rolls back the UPDATE statement with ROLLBACK statement

The result is that U2 has seen data that was never committed. In other words, he saw data that never been existed. U2 has seen dirty data.

- Non-repeatable Read or Non-reproducible Read
Non-repeatable Read is a special version of dirty read (or inconsistent read). Here a user reads partly dirty and partly clean data and combines it. The user is not aware of the fact that this result is based upon data that is only partly clean.

1. User U1 retrieves all players resident in Stratford and writes their player numbers on a piece of paper:

```
SELECT PLAYERNO
FROM PLAYERS
WHERE TOWN = 'Stratford';
```

The result is 6, 83, 2, 7, 57, 39, and 100. Then U1 starts a new transaction.

2. A few seconds later, user U2 changes the address of player 7 who lives in Stratford

```
UPDATE PLAYERS
SET TOWN = 'Eltham'
WHERE PLAYERNO = 7;
```

3. Next, user U2 ends the transaction with a COMMIT

4. U1 queries

```
SELECT PLAYERNO, NAME, INITIALS,
STREET, HOUSENO, POSTCODE, TOWN
FROM PLAYERS
WHERE PLAYERNO IN (6, 83, 2, 7, 57, 39, 100);
```

The second SELECT statement in the same transaction does not give the same picture of the database. The result of the first SELECT statement cannot be reproduced, which is not desirable.

- Phantom Read

1. User U1 looks for all players residing in Stratford.

```
SELECT PLAYERNO
FROM PLAYERS
WHERE TOWN = 'Stratford';
```

2. Sometime later, U2 adds a new player who lives in Stratford and ends the transaction with COMMIT.

3. U1 sees one more row when he executes the same SELECT statement: the row U2 entered.

- Lost Update

Change made by a user overwrites the change of another user.

1. U1 wants to increase the amount of the penalty with payment number 4 by $25.
UPDATE PENALTIES
SET AMOUNT = AMOUNT + 25
WHERE PAYMENTNO = 4;

2. Next, U1 ends the transaction with a COMMIT
3. U2 executes

UPDATE PENALTIES
SET AMOUNT = AMOUNT + 30
WHERE PAYMENTNO = 4;

4. U2 ends the transaction with COMMIT.