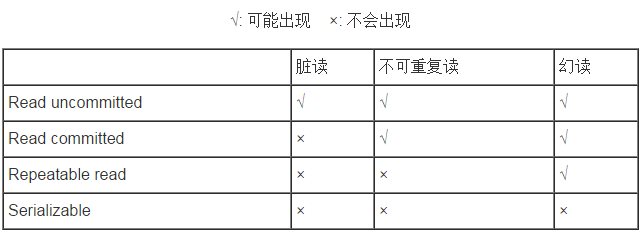
[**数据库事务4种隔离级别及7种传播行为**](https://www.cnblogs.com/WJ-163/p/6023054.html)

一、隔离级别：

数据库事务的隔离级别有4个，由低到高依次为Read uncommitted、Read committed、Repeatable read、Serializable，这四个级别可以逐个解决脏读、不可重复读、幻读这几类问题。



１. ISOLATION\_READ\_UNCOMMITTED：这是事务最低的隔离级别，它充许令外一个事务可以看到这个事务未提交的数据。  
      这种隔离级别会产生脏读，不可重复读和幻像读。  
２. ISOLATION\_READ\_COMMITTED：保证一个事务修改的数据提交后才能被另外一个事务读取。另外一个事务不能读取该事务未提交的数据  
３. ISOLATION\_REPEATABLE\_READ：这种事务隔离级别可以防止脏读，不可重复读。但是可能出现幻像读。  
      它除了保证一个事务不能读取另一个事务未提交的数据外，还保证了避免下面的情况产生(不可重复读)。  
４. ISOLATION\_SERIALIZABLE：这是花费最高代价但是最可靠的事务隔离级别。事务被处理为顺序执行。  
      除了防止脏读，不可重复读外，还避免了幻像读。

我们使用 test 数据库，新建 tx 表：－－－ＭｙＳＱＬ数据库

**第1级别：Read Uncommitted(读取未提交内容)**  
(1)所有事务都可以看到其他未提交事务的执行结果  
(2)本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少  
(3)该级别引发的问题是——脏读(Dirty Read)：读取到了未提交的数据

[复制代码](javascript:void(0);)

#首先，修改隔离级别

set tx\_isolation='READ-UNCOMMITTED';

select @@tx\_isolation;

+------------------+

| @@tx\_isolation |

+------------------+

| READ-UNCOMMITTED |

+------------------+

#事务A：启动一个事务

start transaction;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务B：也启动一个事务(那么两个事务交叉了)

在事务B中执行更新语句，且不提交

start transaction;

update tx set num=10 where id=1;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 10 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务A：那么这时候事务A能看到这个更新了的数据吗?

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 10 | --->可以看到！说明我们读到了事务B还没有提交的数据

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务B：事务B回滚,仍然未提交

rollback;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务A：在事务A里面看到的也是B没有提交的数据

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 | --->脏读意味着我在这个事务中(A中)，事务B虽然没有提交，但它任何一条数据变化，我都可以看到！

| 2 | 2 |

| 3 | 3 |

+------+------+

[复制代码](javascript:void(0);)

**第2级别：Read Committed(读取提交内容)**

(1)这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）  
(2)它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变  
(3)这种隔离级别出现的问题是——不可重复读(Nonrepeatable Read)：不可重复读意味着我们在同一个事务中执行完全相同的select语句时可能看到不一样的结果。  
     |——>导致这种情况的原因可能有：(1)有一个交叉的事务有新的commit，导致了数据的改变;(2)一个数据库被多个实例操作时,同一事务的其他实例在该实例处理其间可能会有新的commit

[复制代码](javascript:void(0);)

#首先修改隔离级别

set tx\_isolation='read-committed';

select @@tx\_isolation;

+----------------+

| @@tx\_isolation |

+----------------+

| READ-COMMITTED |

+----------------+

#事务A：启动一个事务

start transaction;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务B：也启动一个事务(那么两个事务交叉了)

在这事务中更新数据，且未提交

start transaction;

update tx set num=10 where id=1;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 10 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务A：这个时候我们在事务A中能看到数据的变化吗?

select \* from tx; --------------->

+------+------+ |

| id | num | |

+------+------+ |

| 1 | 1 |--->并不能看到！ |

| 2 | 2 | |

| 3 | 3 | |

+------+------+ |——>相同的select语句，结果却不一样

|

#事务B：如果提交了事务B呢? |

commit; |

|

#事务A: |

select \* from tx; --------------->

+------+------+

| id | num |

+------+------+

| 1 | 10 |--->因为事务B已经提交了，所以在A中我们看到了数据变化

| 2 | 2 |

| 3 | 3 |

+------+------+

[复制代码](javascript:void(0);)

**第3级别：Repeatable Read(可重读)**  
(1)这是MySQL的默认事务隔离级别  
(2)它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行  
(3)此级别可能出现的问题——幻读(Phantom Read)：当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行  
(4)InnoDB和Falcon存储引擎通过多版本并发控制(MVCC，Multiversion Concurrency Control)机制解决了该问题

[复制代码](javascript:void(0);)

#首先，更改隔离级别

set tx\_isolation='repeatable-read';

select @@tx\_isolation;

+-----------------+

| @@tx\_isolation |

+-----------------+

| REPEATABLE-READ |

+-----------------+

#事务A：启动一个事务

start transaction;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务B：开启一个新事务(那么这两个事务交叉了)

在事务B中更新数据，并提交

start transaction;

update tx set num=10 where id=1;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 10 |

| 2 | 2 |

| 3 | 3 |

+------+------+

commit;

#事务A：这时候即使事务B已经提交了,但A能不能看到数据变化？

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 1 | --->还是看不到的！(这个级别2不一样，也说明级别3解决了不可重复读问题)

| 2 | 2 |

| 3 | 3 |

+------+------+

#事务A：只有当事务A也提交了，它才能够看到数据变化

commit;

select \* from tx;

+------+------+

| id | num |

+------+------+

| 1 | 10 |

| 2 | 2 |

| 3 | 3 |

+------+------+

[复制代码](javascript:void(0);)

**第4级别：Serializable(可串行化)**  
(1)这是最高的隔离级别  
(2)它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之,它是在每个读的数据行上加上共享锁。  
(3)在这个级别，可能导致大量的超时现象和锁竞争

[复制代码](javascript:void(0);)

#首先修改隔离界别

set tx\_isolation='serializable';

select @@tx\_isolation;

+----------------+

| @@tx\_isolation |

+----------------+

| SERIALIZABLE |

+----------------+

#事务A：开启一个新事务

start transaction;

#事务B：在A没有commit之前，这个交叉事务是不能更改数据的

start transaction;

insert tx values('4','4');

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

update tx set num=10 where id=1;

ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

[复制代码](javascript:void(0);)

二、传播行为

1、PROPAGATION\_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

2、PROPAGATION\_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。‘

3、PROPAGATION\_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

4、PROPAGATION\_REQUIRES\_NEW：创建新事务，无论当前存不存在事务，都创建新事务。

5、PROPAGATION\_NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

6、PROPAGATION\_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

7、PROPAGATION\_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION\_REQUIRED类似的操作。