什么是MVCC？（要自己具体了解）

MVCC是一种多版本并发控制机制。

MVCC是为了解决什么问题？

1.大多数的mysql事务型存储，如，InnoDB，Falcon以及PBXT都不使用一种简单的行锁机制。事实上，他们都和mvcc-多版本并发控制来一起使用。

2.大家都应该知道，锁机制可以控制并发操作，但是其系统开销较大，而mvcc可以在大多数情况下代替行级锁，使用mvcc，能降低其系统开销。

MVCC的实现：

MVCC是通过保存数据在某个时间点的快照来实现的。不同存储引擎的MVCC实现是不同的，典型的有乐观并发控制和悲观并发控制。

MVCC的具体实现分析：

下面，我们通过InnoDB的MVCC实现来分析MVCC是怎样进行并发控制的。

InnoDB的MVCC，是通过在每行记录后面保存两个隐藏的列来实现的，这两个列，分别保存了这个行的创建时间，一个保存的是行的删除时间。这里存储的并不是实际的时间值，而是版本号（可以理解为事务的ID），每开始一个新的事务，系统版本号就会自动递增，事务开始时刻的系统版本号会作为事务的ID，下面看一在REPEATBALE READ隔离级别下，MVCC具体是如何操作的。

来一个简单的例子：

```
create table yang(
id int primary key auto_increment,
name varchar(20));
```

> 假设系统的版本号从1开始.

INSERT

InnoDB为新插入的每一行保存当前系统版本号作为版本号.
第一个事务ID为1;

```
start transaction;
insert into yang values(NULL,'yang') ;
insert into yang values(NULL,'long');
insert into yang values(NULL,'fei');
commit;
```

对应在数据中的表如下(后面两列是隐藏列,我们通过查询语句并看不到)

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	undefined
2	long	1	undefined
3	fei	1	undefined

当进行查询时（InnoDB引擎）：
InnoDB会根据一下两个条件检查每行记录：
a. InnoDB只会查找版本早于当前事务版本的数据行（也就是，行的系统版本号小于或等于事务的系统版本号），这样可以确保事务读取的行，要么是在事务开始前已经存在的，要么是事务自身插入或者修改过的。
b. 行的删除版本要么未定义，要么大于当前事务的版本号，这可以确保事务读取到的行，在事务开始之前未被删除。
只有a，b同时满足的记录，才能返回作为查询结果。

DELETE
InnoDB会为删除的每一行保存当前系统的版本号（事务ID）作为删除标识

看下面的具体例子分析:

第二个事务,ID为2;

```
1  start transaction;
2  select * from yang;   //(1)
3  select * from yang;   //(2)
4  commit;
```

假设1:

假设在执行这个事务ID为2的过程中，刚执行到(1)，这时，有另一个事务ID为3往这个表里插入了一条数据，这个事务ID为3

```
1  start transaction;
2  insert into yang values(NULL,'tian');
3  commit;
```

这时表中的数据如下：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	undefined
2	long	1	undefined
3	fei	1	undefined
4	tian	3	undefined

然后接着执行事务2中的(2),由于id=4的数据创建时间(事务id为3)，执行当前事务的ID为2，而InnoDB只会查找事务ID小于等于当前事务ID的数据行，所以id=4的数据行并不会在执行事务2中的(2)被检索出来，在事务2中的两条select语句检索出来的数据都只会下表：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	undefined
2	long	1	undefined
3	fei	1	undefined

假设2:

假设在执行这个事务ID为2的过程中，刚执行到(1)，假设事务执行完事务3后，接着又执行了事务4，第4个事务：

```
start  transaction;
delete from yang where id=1;
commit;
```

此时数据库中的表如下：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	4
2	long	1	undefined
3	fei	1	undefined
4	tian	3	undefined

接着执行事务ID为2的事务(2)，根据select检索条件可以知道，它会检索创建时间(创建事务的ID)小于当前事务ID的行和删除时间(删除事务的ID)大于当前事务的行，而id=4的行上面已经说过，而id=1的行由于删除时间(删除事务的id)大于当前事务id，所以事务2的(2)select * from yang也会把id=1的数据检索出来。所以，事务2中的两条select语句检索出来的数据都如下：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	4
2	long	1	undefined
3	fei	1	undefined

UPDATE：

InnoDB执行Update，实际上是新插入了一行记录，并保存其创建时间为当前事务id，同时保存当前事务id到要update的行的删除时间。

假设3:

假设在执行完事务2的(1)后又执行，其他用户执行了事务3，4.这时，又有一个用户对这张表执行了update操作：

第5个事务：

```
start  transaction;
update yang set name='Long' where id=2;
commit;
```

根据update的更新原则:会生成新的一行，并在原来要修改的列的删除时间列上添加本事务id，得到表如下：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	4
2	long	1	5
3	fei	1	undefined
4	tian	3	undefined
2	Long	5	undefined

继续执行事务2的(2),根据select语句的检索条件，得到下表：

id	name	创建时间(事务ID)	删除时间(事务ID)
1	yang	1	4
2	long	1	5
3	fei	1	undefined

还是和事务2中(1)select 得到相同的结果.