# MVCC知识点总结

为保持数据一致性，最简单的做法就是加锁。但加锁对性能损耗太大，尤其是对于读操作多于写操作的系统来说，给读操作也加锁，对性能来说更是无谓的浪费。

但如果不加锁，就很容易产生幻读。

为解决以上两种情况，就采用MVCC（多版本并发控制），简单地说就是事务读取的是数据快照。

InnoDB的实现方式是在每条记录后增加两个隐藏的字段“创建版本号”、“删除版本号”，版本号随着事务的创建而递增。

INSERT：

为每一个新增的记录添加当前系统版本号为创建版本号（假设为1）

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | CREATE\_VERSIONNO | DEL\_VERSIONNO |
| 1 | LEO | 1 | - |
| 2 | NEO | 1 | - |

DELETE：

删除行的“删除版本号”设为当前事务版本号（假设为2）

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | CREATE\_VERSIONNO | DEL\_VERSIONNO |
| 1 | LEO | 1 | 2 |
| 2 | NEO | 1 | - |

SELECT：

假设当前事务版本号为3，查询数据需要同时满足两个条件：

创建版本号<=3（说明要查的是本事务之前创建的数据）

删除版本号>3 或者 删除版本号为NULL（事务版本号小于本事务版本号，就是以前事务已经删除了，本次事务查不出来。但是在并发中，有事务4删除了数据，删除版本号为4，对于事务3来说却是没有删除的，所以要查出来。）

由上可得查询的结果应该是：

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | CREATE\_VERSIONNO | DEL\_VERSIONNO |
| ~~1~~ | ~~LEO~~ | ~~1~~ | ~~2~~ |
| 2 | NEO | 1 | - |

UPDATE：

假设当前事务版本号为5，执行两个操作：

旧记录“删除版本号”设为5。

插入一条新纪录，“创建版本号”为5。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | CREATE\_VERSIONNO | DEL\_VERSIONNO |
| ~~1~~ | ~~LEO~~ | ~~1~~ | ~~2~~ |
| ~~2~~ | ~~NEO~~ | ~~1~~ | ~~5~~ |
| 2 | NEO2 | 5 | - |

如此，事务版本号为6的操作再查询的时候，就只会查出NEO2这条记录了。

MVCC的缺点就是为了实现多版本，不得不额外维护隐藏的字段，以及维护冗余的记录。