Mysql学习

1. 视图

视图本身是不存储数据的，只是保存了一系列的经常使用SQL查询语句，当用户每次使用视图时，都会实时地执行保存的SQL查询语句，获得结果。因此，每次使用视图时，获得的都是最新的数据。

有些时候视图还会被用于限制用户对普通表的查询操作，对于这类用户只赋予对应视图的select操作权限，仅让他们只能读取特定的行或列的数据。

2. 复合索引

多个字段组成的索引，比如：(1,1), (1,2), (2,1), (2,2), (2,3)……

因此，在创建复合索引时，指定的索引字段的顺序，以及查询时，条件字段的顺序，都会影响是否可以用到复合索引。比如在上例中，如果where中只给出第二个字段的条件，就无法使用复合索引。

如果在条件查询中，没有完全按照顺序查询，则会使用到索引的一部分：

比如index(A,B,C)，where A=xx and C=xx，此时会用到复合索引的A部分

3. 存储过程

相当于数据库内存存储的函数。create procedure xxx()

创建函数时，使用in标识输入参数，使用out标识输出参数，函数的输出参数会存储在数据库中，作为一个可以访问的全局变量。

使用IF, ELSE IF ,ELSE来控制条件

使用REPEAT， END REPEAT和WHILE来进行循环操作

使用declare来声明局部变量

使用SET操作变量，SET N=0; SET N=N+1….，定义变量的作用域GLOBAL/SESSION，@var定义用户变量

使用declare condition和delcare handler for用于异常处理

4. 存储函数与存储过程的区别

存储过程可以有多个in,out,inout参数，而存储函数只有输入参数类型，而且不能带in

存储过程实现的功能要复杂一些；而存储函数的单一功能性(针对性)更强。

存储过程可以返回多个值；存储函数只能有一个返回值。

存储过程一般独立的来执行；而存储函数可以作为其他SQL语句的组成部分来出现。

存储过程可以调用存储函数。但函数不能调用存储过程。

5. 触发器

定义当某些事件触发时，数据库执行特定的操作。

6. 排它锁和共享锁

当数据对象被加上排它锁时，其他的事务不能对它读取和修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。数据库利用这两 种基本的锁类型来对数据库的事务进行并发控制。

共享锁（S锁）又称读锁，若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

排它锁（X所）又称写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，直到T释放A上的锁。这保证了其他事务在T释放A上的锁之前不能再读取和修改A。

7. 乐观锁与悲观锁

乐观锁对共享数据的访问冲突持乐观态度，一般没有数据库默认的实现，可以通过给表添加version字段来实现。

悲观锁对共享数据的访问冲突持保守态度，一般可以通过数据库自身提供的锁机制(共享锁与排它锁)来实现，保证强烈的排他性与独占性，但是对数据库的性能可能会有很大的开销。

7. 死锁的产生原因

第一种情况：一个用户A 访问表A(锁住了表A),然后又访问表B；另一个用户B 访问表B(锁住了表B)，然后企图访问表A，这就死锁就产生了。

这种情况比较常见，可以通过修改程序来解决，在访问多个表时，一定要按照相同的顺序来访问，如果大家都是先访问表A，再访问表B，就不会产生这种情况下的死锁了。

……….

8. 不同的数据库引擎

9. mysql事务与锁

表锁：锁住整张表，开销小，速度快

行锁：锁住表中的某一行，最大限度地支持并发处理，但是开销大，速度慢

事务的实现就是通过把表锁、行锁以及读锁、写锁这些机制结合起来，来实现对事务与共享数据的操作。

10. read uncommitted加锁情况

读的时候不申请锁，写的时候申请行级的排它锁（写锁），并在事务结束（commit）以后，释放之前申请的排它锁。

体现实例：关闭mysql默认的自动提交事务功能；事务A对某一行进行update操作，并不提交事务；事务B也对同一行进行update操作，此时事务B会处于lock wait状态阻塞住，通过查看事务A,B的信息，可以看到A申请了这一行的排它锁，而B在等待A释放排它锁；而read uncommitted的特性就是可以读取到未提交的数据，说明在读的时候，read uncommitted是不申请锁的。

11. read committed加锁情况

Read committed的效果是，只能读取到已经commit的数据。

如果只用锁来实现：在read uncommitted的基础上，每次读都申请共享锁（读锁），那么在事务未commit的时候，写锁还没有被释放，读锁的申请就会被阻塞住。但是，这样实现的效率非常低下，因为实际情况下，读操作的数目远远大于写操作，写的时候所有的读都被阻塞住，效率太低。

实际上innodb中read committed的实现：实际上read committed级别下，读的时候也是不会申请共享锁的，也就是说即使事务A中的修改还没有提交，事务B已然能够非阻塞地读到修改之前的值。

关于读到的值的判定：结合undo log和各种事务的时间顺序来决定读到值。如果该行的所有事务，在本事务开始之前就已经commit了，就可以直接读取本行的数据；如果该行的所有事务在本事务开始之后才开始，那么当前的值就是不可见的，需要用过undo log还原出当时的值，再返回给用户；如果本事务位于活跃用户列表中，那么寻找合适的undo log去还原当时的值。

12. repeatable read加锁情况

Repeatable read的效果是：在同一个事务中，多次查询同一条记录，永远会得到相同的结果。

如果只用锁来实现：在查询的时候加行级的共享锁，这样其他事务就申请不到排它锁，也无法对该行记录进行修改；update时如同之前的那些隔离级别，会上排它锁。这样做的问题，还是如同read committed中说到的，效率太低。

实际上innodb中repeatable read的实现： 和read committed的实现类似，在读的时候并不会申请锁，也是一种类似于MVCC的方法，区别在于repeatable read是在开始事务的时候，生成当前活跃事务的列表；而read committed是在语句执行之前生成当前活跃事务的列表。

读不影响写，写不影响读。

13.