第九周作业：数据库问答题

题目01 – 请你说一说MySQL的锁机制

（1）按照锁的粒度，锁的功能来分析

按照锁的粒度，锁能够分为：

- 全局锁：锁住整个数据库，由MySQL的SQLLayer层实现
- 表级锁：锁某张表，有MySQL的SQLLayer层实现
- 行级锁：锁住某行的索引，也可以锁定索引之间的间隙，有存储引擎实现。根据锁的范围，分为：
    - 记录锁：锁定索引中的一条记录
    - 间隙锁：锁住索引记录左右的区间
    - 临键锁：记录锁和间隙锁的组合，解决幻读问题
    - 插入意向锁：做 insert 时添加的对记录id的锁

（2）什么是死锁，为什么会发生，如何排查?

死锁就是一个session1获取了一个锁a，另一个session2获取了另外一个锁b，此时session1又要获取锁b，等待session2释放锁b，而锁b要获取锁a进行操作，此时等待session1释放锁a，session1和session2互相等待对方已获取的资源。

可以通过sql查看最近一次的死锁记录日志：

```SQL
SHOW ENGINE INNODB STATUS;
```

（3）行锁是通过加在什么上完成的锁定?

行锁是通过加在索引上的索引记录来实现的。

- 记录锁
  - 仅仅锁住索引记录的一行，在单条索引记录上加锁。
- 间隙锁
  - 仅仅锁住一个索引区间
  - 在索引记录之间的间隙中加锁，或者是在某一条索引记录之前或者之后加锁，并不包括该索引记录本身。
  - 间隙锁可用于防止幻读
- 临键锁
  - 相当于记录锁 + 间隙锁
  - 默认情况下，innodb使用临键锁来锁定记录。
  - 当查询的索引含有唯一属性的时候，临键锁会被优化，将其降级为记录锁
  - 临键锁在不同的场景中会发生退化
- 插入意向锁

（4）详细说说这条SQL的锁定情况： delete from tt where uid = 666;

- 读已提交RC隔离级别
  - uid列是主键
    - 此时，只需要在uid列为666的记录上加写锁即可
  - uid列是二级唯一索引

        - 此时，需要在二级唯一索引和主键索引uid列为666的位置上增加写锁即可
    - uid列是二级非唯一索引
        - 此时，需要在二级非唯一索引为uid=666的所有记录上与对应的主键索引的记录上增加写锁
    - uid列上没有索引
        - 此时进行全表扫描，因此每条记录上都会加上写锁，但是这样性能就会有所影响，因此MySQL在这个位置做了优化，如果不满足条件记录，索引上就会释放锁
- 可重复读RR隔离级别
    - uid列是主键
        - 此时，只需要在uid列为666的记录上加写锁即可
    - uid列是二级唯一索引
        - 此时，需要在二级唯一索引和主键索引uid列为666的位置上增加写锁即可
    - uid列是二级非唯一索引
        - 此时，在二级非唯一索引对应数据上上增加写锁和间隙锁防止出现幻读，在主键索引上增加写锁
    - uid列上没有索引
        - 此时进行全表扫描，全表记录增加写锁和间隙锁

题目02 – 请你说一说MySQL的SQL优化

Explain

MySQL提供了一个 explain命令， 它可以对 SELECT 语句的执行计划进行分析，并输出 SELECT 执行的详细信息，以供开发人员针对性优化。explain命令用法很简单, 在 SELECT前加上 explain 就行。

- id：SELECT识别符
- select_type：表示单位查询的查询类型
    - simple： 普通查询
    - primary
    - union
    - dependent union

    - subquery：子查询
    - dependent subquery
    - derived：派生表
- table：表示查询的表
- partitions：使用的哪些分区
- type：表示表的连接类型
    - system
    - const
    - eq_ref
    - ref
    - fulltext
    - ref_or_null
    - unique_subquery
    - index_subquery
    - range
    - index_merge
    - index
    - ALL
- possible_keys：此次查询中可能选用的索引
- key：查询真正使用到的索引
- key_len：显示MySQL决定使用的索引
- sizeref：哪个字段或常数与 key一起被使用
- rows：显示此查询一共扫描了多少行，这不是精确的值
- filtered：表示此查询条件所过滤的数据的百分比
- Extra：额外信息
    - Using filesort：使用文件排序，说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取
    - Using index：表示相应的SELECT查询中使用到了索引
    - Using where：表示MySQL将对InnoDB提取的结果在SQL Layer层进行过滤，过滤条件字段无索引
    - Using join buffer：表明使用了连接缓存

索引优化建议

1. 表记录很少不需创建索引
2. 一个表的索引个数不能过多
3. 频繁更新的字段不建议作为索引
4. 区分度低的字段，不建议建索引
5. 在InnoDB存储引擎中，主键索引建议使用自增的长整型，避免使用很长的字段：
6. 不建议用无序的值作为索引
7. 尽量创建组合索引，而不是单列索引

LIMIT优化

LIMIT的优化问题是 offset 的问题，它会导致 MySQL 扫描大量不需要的行然后再抛弃掉。

解决方案：单表分页时，使用自增主键排序之后，先使用 where 条件 id > offset值，limit后面只写rows

子查询优化

可以使用连接查询（JOIN）代替子查询，连接查询时不需要建立临时表，其速度比子查询快。

其他查询优化

- 小表驱动大表：建议使用left join时，以小表关联大表，JOIN两张表的关联字段最好都建立索引并且字段类型一致
- 避免全表扫描：注意索引失效，避免索引失效导致全表扫描
- 避免MySQL放弃索引：如果MySQL估计使用全表扫描要比使用索引快，则不使用索引。
- WHERE条件中尽量不要使用notin语句，建议使用not exists