数据库优化

题目 01- 请你说一说 MySQL 的锁机制

一、锁分类

按功能分类：

1. **共享锁(读锁)**：允许同一个数据被加多个读锁，读取相互不阻塞，但是无法再被添加写锁，不允许其他事务修改当前读锁所保护的数据。加锁方式 'select...lock in share mode'
2. **排他锁(写锁)**：当一个数据被加了写锁，其他事务就不能对其上任何锁，不支持任何其他的读取与修改。加锁方式 'select...for update'

按锁粒度分类：

1. **全局锁**：针对整个数据库执行全局层面的锁，上锁后任何对该数据库的DDL、DML语句都将无法执行，只能进行查询操作。一般使用场景是用于数据库备份。但不建议使用该方法，因为阻塞所有非读操作会影响业务的正常执行。因此InnoDB下执行全库备份时，可以在mysqldump指令后使用--single-transaction参数，利用 mvcc提供一致性视图保证数据一致性，不使用全局锁。
2. **表级锁**：

   表级锁是InnoDB存储引擎外的概念，他是在SQL Layer中实现的。从外层对整个表进行锁定。后续会讲到当进行全表扫描时，此时加锁会将整个表的所有数据(RR级别下还有间隙)全部都加锁，虽然他们的表现形式都是整个表的数据都被锁了，但是这应该是两个概念。表级锁包含如下几个类型：
   - **表读锁**：某个session为表设置表读锁后，其他session只能查询该表，不能对该表进行更新操作。但是不同的是，为该表上读锁的session，也不能对该表进行更新操作。可以理解为该表上读锁后，对任何session事务都变为只读。
   - **表写锁**：与普通写锁概念一致，为某表上写锁后，之后当前session可对其进行增删改查。其余session的增删改查操作将被阻塞。
   - **元数据锁**：元数据锁的作用是保证DDL和DML语句不会产生冲突。当我们对某个表进行增删改查操作时，mysql会自动为该表上元数据读锁。此时只允许查询该表结构，不能修改该表结构。防止数据操作执行过程中，表结构变化产生冲突。当session对表结构进行修改时，musql会自动为该表上元数据写锁。
   - **自增锁**：发生字段数据自增时，为自增数据自动上的锁，防止并发安全问题。
3. **行级锁**：

   行级锁是由存储引擎来实现的，InnoDB中为数据上锁是通过对索引上的索引项加锁来实现的。只有通过索引条件检索的数据， InnoDB才使用行级锁，否则InnoDB会锁住该表的所有数据，实际上就是表锁。对于UPDATE、DELETE、INSERT，InnoDB会自动为数据添加写锁，但当执行SELECT操作时，MVCC框架下是默认不加锁的，因此需要人为地去为其加上读锁或写锁。

行锁包含如下几个类型：

- **记录锁**：只锁住某条涉及的记录。一般只有在我们通过主键进行等值查询，只会查询出单条记录时，才会在聚簇索引上的该数据添加记录锁。或者是针对辅助索引进行等值条件查询，按查询条件锁住辅助索引的同时，也会在聚簇索引上添加一个或多个记录锁。总之单条记录上锁就是记录锁。
- **间隙锁**：锁住某两个索引之间的间隙，是一个范围性的索引，两侧是开区间，不包含两侧记录。这个是RR隔离级别下避免幻读的根本原理。可以理解为当我们查询某个索引，但是未命中时，我们需要锁住该区域。因为我们现在查询出来该区间是存在符合条件的数据的，锁住该区域放止在我们执行下一步操作之前该区域出现别事务创建的新纪录，从而出现幻读现象。此外辅助索引因为其和重复性，所以在等值条件命中/未命中/范围查询等情况下都要加间隙锁，防止该范围出现新数据导致一致性冲突。
- **临键锁**：相当于记录锁+临键锁，是一个左开右闭的区间。当我们进行范围查询时，如果命中了主键索引或是辅助索引，那么该索引树上的数据就需要加临键锁，锁住包含命中以及命中数据之间的数据空间。
- **插入意向锁**：可以理解为一种在执行插入语句时，为插入数据所处的间隙上的一个间隙锁。但该锁不会锁住整个间隙，其他事务也可以向这个间隙插入数据。但是插入意向锁会为当前间隙中新插入的数据添加一个记录锁，防止有重复的数据在相同的位置进行二次插入。

4.3 加锁规则【非常重要】

主键索引

- 等值条件，命中，加记录锁
- 等值条件，未命中，加间隙锁
- 范围条件，命中，包含where条件的临键区间，加临键锁
- 范围条件，没有命中，加间隙锁

辅助索引

- 等值条件，命中，命中记录的辅助索引项 + 主键索引项加记录锁，辅助索引项两侧加间隙锁
- 等值条件，未命中，加间隙锁
- 范围条件，命中，包含where条件的临键区间加临键锁。命中记录的id索引项加记录锁
- 范围条件，没有命中，加间隙锁

4. **意向锁**：核心作用就是提升行锁与表锁相互合作的效率。当某个事务为数据表添加行锁后，Mysql会根据其行锁类型自动为该表添加意向读锁/意向行锁。此时当其他事务想再添加表级写锁时，可直接通过已有的意向锁类型快速判断出是否可以添加。

举个栗子：

事务A修改user表的记录r，会给记录r上一把行级的**写锁**，同时会给user表上一把**意向写锁（IX）**，这时事务B要给user表上一个表级的**写锁**就会被阻塞。**意向锁**通过这种方式实现了行锁和表锁共存，且满足事务隔离性的要求。

当我们需要加一个写锁时，需要根据意向锁去判断表中有没有数据行被锁定；

（1）如果行锁，则需要遍历每一行数据去确认；

（2）如果表锁，则只需要判断一次即可知道有没数据行被锁定，提升性能。

二、详细说说这条 SQL 的锁定情况： delete from tt where uid = 666 ;

1. **RC隔离级别下**：

- **uid为主键时**：直接在聚簇索引上，uid=666的数据位置添加写锁。
- **uid为唯一性辅助索引时**：在辅助索引上，uid=666的数据位置添加写锁。此外需找到对应主键id，在聚簇索引上相应位置添加第二个写锁。(聚簇索引上也添加写锁，防止其他事务不通过辅助索引直接通过聚簇索引修改该被锁定的数据)
- **uid为非唯一性辅助索引时**：在辅助索引上，为所有uid=666的数据位置都添加写锁。再找到所有数据对应主键id，在聚簇索引上相应的所有位置添加写锁。
- **uid不为索引时**：走全表扫描，为聚簇索引上的所有数据全部上锁。因为全表扫描与索引扫描相比是一个十分耗时的操作，如果全表扫描不上锁，极有可能出现扫描过程中数据被修改的现象。因此全表扫描模式下，Mysql会默认锁住所有数据。但Mysql在此基础上做出了优化，对于已检查的不满足条件的记录，会在判断后放锁，最终持有的，是满足条件的记录上的锁，但是不满足条件的记录上的加锁/放锁动作不会省略。

2. **RR隔离级别下**：RR隔离级别与RC隔离级别最大的区别就是引入了间隙锁，解决了幻读的问题。

- **uid为主键/唯一性索引时**：与RC隔离级别相同
- **uid为非唯一性索引时**：为防止幻读问题，RR隔离级别下，会将辅助索引中所有命中筛选条件的数据四周间隙都加上间隙锁。同样还需为聚簇索引上的相应命中的数据都添加上数据锁。
- **uid不为索引时**：执行全表扫描，会为聚簇索引上的所有数据添加记录锁，所有间隙添加间隙锁，相当于为该表上了表级锁，对数据无法进行任何修改和新增删除。

三、什么是死锁，为什么会发生，如何排查？

死锁就是两个事务互相拥有对方所需要的数据锁，但又在等待对方释放自己所需要的锁，因此两周都能更近一步执行后续操作，导致死锁。如果需要排插死锁，可执行下述语句，查看innodb引擎时间信息，找到'LATEST DETECTED DEADLOCK'关键词，分析死锁原因。

```
SHOW ENGINE INNODB STATUS;
```

6.2 如何避免死锁呢?

MySQL默认会主动探知死锁，并回滚某一个影响最小的事务。等另一事务执行完成之后，再重新执行该事务。

1. 注意程序的逻辑：根本的原因是程序逻辑的顺序交叠，最常见的是交差更新
2. 保持事务的轻量：越是轻量的事务，占有越少的锁资源，这样发生死锁的几率就越小
3. 提高运行的速度：避免使用子查询，尽量使用主键等等
4. 尽量快提交事务，减少持有锁的时间：越早提交事务，锁就越早释放

题目 02- 请你说一说 MySQL 的 SQL 优化

1. 数据库调优调什么？什么影响数据库性能?

- 调整索引设计、创建
- 优化SQL语句，寻求最优、性能最好的写法【巧用索引】
- 调整数据库表结构
- 调整MySQL配置：**最大连接数，连接超时**，线程缓存，查询缓存，排序缓存，连接查询缓存...
- 调MySQL宿主机OS：TCP连接数，打开文件数，线程栈大小...
- 调服务器硬件：更多核CPU、更大内存

2. 如何调优?

- 使用Jmeter对数据库进行压力测试，调整MySQL连接池配置，例如**MaxWait**、**MaxActive**等参数。
- 使用Explain对SQL语句进行执行计划查询，通过type字段判断SQL语句的执行性能
- 使用Profile对SQL语句进行详细分析，可以定位出一条SQL语句执行的各种资源消耗情况，比如CPU，IO等，以及该SQL执行所耗 费的时间等。
- 慢日志查询：通过打开慢日志查询功能，使用mysqldumpslow工具对指定SQL语句进行慢查询检测与分析
- 数据库结构优化：对于字段较多的表，将部分使用频率较低的表分离出来形成新表【分表】。对于经常需要JOIN连表查询的数据，可单独创建一张中间表专门用于连表查询结果直接返回。增加冗余字段，减少连表查询。
- 服务器层面优化：设置足够大的 innodb_buffer_pool_size ，将数据读取到内存中，推荐设置为物理内存的 50%~80%。减少不必要的日志IO，对于生产环境来说，很多日志是不需要开启的，比如：通用查询日志、慢查询日志、错误日志。
- 硬件优化