mysql 优化 面试题

mysql 优化

1、MYSQL 优化主要分为以下四大方面：
设计：存储引擎，字段类型，范式与逆范式
功能：索引，缓存，分区分表。
架构：主从复制，读写分离，负载均衡。
合理 SQL：测试，经验。

优先考虑的是表结构、选择合适的字段、索引优化、结合 Redis 缓存、主从分离、（无可奈何才用 分区、分表、分库）

mysql 保存的数据格式是什么？

```
安装 mysql 时选择的存储引擎是 MYISAM 的，则数据存储在 .MYD 文件中；
选择的是 innodb 存储引擎，则数据是统一存储在一个叫 ibdata1 的文件中的
```

mysql 如何编译成我们能识别的数据格式？

mysql 索引在内存中以什么格式保存？

B+树是一个怎样的树状？为什么会这样？

什么是事务？

锁，行锁，表锁，间隙锁，幻读、脏读，重复读？

MySQL 锁可以按使用方式分为：乐观锁与悲观锁。按粒度分可以分为表级锁，行级锁，页级锁。

表锁

从锁的粒度，我们可以分成两大类：

```
表锁 :开销小，加锁快；不会出现死锁；锁定力度大，发生锁冲突概率高，并发度最低。
行锁:开销大，加锁慢；会出现死锁；锁定粒度小，发生锁冲突的概率低，并发度高 不同的存储引擎支持的锁粒度是不一样的。*    InnoDB 行锁和表锁都支持、MyISAM 只支持表锁！
*    InnoDB 只有通过索引条件检索数据才使用行级锁，否则，InnoDB 使用表锁也就是说，InnoDB 的行锁是基于索引的！
```

表锁下又分为两种模式：表读锁（Table Read Lock）&& 表写锁（Table Write Lock）
从下图可以清晰看到，在表读锁和表写锁的环境下：**读读不阻塞，读写阻塞，写写阻塞！**

读读不阻塞：当前用户在读数据，其他的用户也在读数据，不会加锁

读写阻塞：当前用户在读数据，其他的用户不能修改当前用户读的数据，会加锁！

写写阻塞：当前用户在修改数据，其他的用户不能修改当前用户正在修改的数据，会加锁！

请求锁模式 是否兼容 当前锁模式	None	读锁	
读锁	是	是	
写锁	是	否	

从上面已经看到了：读锁和写锁是互斥的，读写操作是串行。

- 如果某个进程想要获取读锁，同时另外一个进程想要获取写锁。在 mysql 中，写锁是优先于读锁的！
- 写锁和读锁优先级的问题是可以通过参数调节的：max_write_lock_count 和 low-priority-updates

行锁

InnoDB 和 MyISAM 有两个本质的区别：InnoDB 支持行锁、InnoDB 支持事务。

InnoDB 实现了以下两种类型的行锁：

- **共享锁（S 锁、读锁）：**允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。即多个客户可以同时读取同一个资源，但不允许其他客户修改。
- **排他锁（X 锁、写锁)：**允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的读锁和写锁。写锁是排他的，写锁会阻塞其他的写锁和读锁。

另外，为了允许行锁和表锁共存，实现多粒度锁机制，InnoDB 还有两种内部使用的意向锁（Intention Locks），这两种意向锁都是表锁：

- **意向共享锁（IS）：**事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的 IS 锁。
- **意向排他锁（IX）：**事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的 IX 锁。

- 意向锁也是数据库隐式帮我们做了，不需要程序员关心！

死锁

1、产生原因

所谓死锁<DeadLock>：是指两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用，它们都将无法推进下去.此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。表级锁不会产生死锁.所以解决死锁主要还是针对于最常用的 InnoDB。

死锁的关键在于：两个(或以上)的 Session 加锁的顺序不一致。

那么对应的解决死锁问题的关键就是：让不同的 session 加锁有次序

2、产生示例

需求：将投资的钱拆成几份随机分配给借款人。
起初业务程序思路是这样的：
投资人投资后，将金额随机分为几份，然后随机从借款人表里面选几个，然后通过一条条 select for update 去更新借款人表里面的余额等。

例如：两个用户同时投资，A 用户金额随机分为 2 份，分给借款人 1，2

B 用户金额随机分为 2 份，分给借款人 2，1，由于加锁的顺序不一样，死锁当然很快就出现了。

对于这个问题的改进很简单，直接把所有分配到的借款人直接一次锁住就行了。

```
Select * from xxx where id in (xx,xx,xx) for update
```

在 in 里面的列表值 mysql 是会自动从小到大排序，加锁也是一条条从小到大加的锁

MVCC 行级锁

MVCC(Multi-Version ConcurrencyControl)多版本并发控制，可以简单地认为：MVCC 就是行级锁的一个变种(升级版)。

在表锁中我们读写是阻塞的，基于提升并发性能的考虑，MVCC 一般读写是不阻塞的(很多情况下避免了加锁的操作)。

可以简单的理解为：对数据库的任何修改的提交都不会直接覆盖之前的数据，而是产生一个新的版本与老版本共存，使得读取时可以完全不加锁。

悲观锁

我们使用悲观锁的话其实很简单(手动加行锁就行了)：select * from xxxx for update，在 select 语句后边加了 for update 相当于加了排它锁(写锁)，加了写锁以后，其他事务就不能对它修改了！需要等待当前事务修改完之后才可以修改.也就是说，如果操作 1 使用 select ... for update，操作 2 就无法对该条记录修改了，即可避免更新丢失。

乐观锁

乐观锁不是数据库层面上的锁，需要用户手动去加的锁。一般我们在数据库表中添加一个版本字段 version 来实现，例如操作 1 和操作 2 在更新 User 表的时，执行语句如下：

update A set Name=lisi,version=version+1 where ID=#{id} and version=#{version}，

此时即可避免更新丢失。

事务的并发？事务隔离级别，每个级别会引发什么问题，MySQL 默认是哪个级别？

脏读是指在一个事务处理过程中读取了另一个事务未提交的数据。
不可重复读：对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值
幻读：事务非独立执行时发生的一种现象，即在一个事务读的过程中，另外一个事务可能插入了新数据记录，影响了该事务读的结果

MySQL 的**默认隔离级别**就是 Repeatable read，可重复读。

从理论上来说, 事务应该彼此完全隔离, 以避免并发事务所导致的问题，然而, 那样会对性能产生极大的影响, 因为事务必须按顺序运行，**在实际开发中，为了提升性能，事务会以较低的隔离级别运行， 事务的隔离级别可以通过隔离事务属性指定。事务的并发问题**
事务的并发问题
1、脏读：事务 A 读取了事务 B 更新的数据，然后 B 回滚操作，那么 A 读取到的数据是脏数据
2、不可重复读：事务 A 多次读取同一数据，事务 B 在事务 A 多次读取的过程中，对数据作了更新并提交，导致事务 A 多次读取同一数据时，结果因此本事务先后两次读到的数据结果会不一致。
3、幻读：可重复读的隔离级别解决了不可重复读的问题，保证了同一个事务里，查询的结果都是事务开始时的状态（一致性）。

小结：不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表。

事务的隔离级别

隔离级别	脏读	不可重复读
读未提交（Read uncommitted）	V	V
读已提交（Read committed）	X	V
可重复读（Repeatable read）	X	X
可串行化（Serializable）	X	X

* 读未提交：另一个事务修改了数据，但尚未提交，而本事务中的 SELECT 会读到这些未被提交的数据脏读
*
不可重复读：事务 A 多次读取同一数据，事务 B 在事务 A 多次读取的过程中，对数据作了更新并提交，导致事务 A 多次读取同一数据时，结果因此本事务先后两次读到的数据结果会不一致。
*
可重复读：在同一个事务里，SELECT 的结果是事务开始时时间点的状态，因此，同样的 SELECT 操作读到的结果会是一致的。但是，会有幻读现象
*
串行化：最高的隔离级别，在这个隔离级别下，不会产生任何异常。并发的事务，就像事务是在一个个按照顺序执行一样。

读未提交（Read uncommitted），就是一个事务能够看到其他事务尚未提交的修改，允许脏读出现.
读已提交（Read committed），事务能够看到的数据都是其他事务已经提交的修改，也就是保证不会看到任何中间性状态，当然脏读也不会出现。
可重复读（Repeatable reads），保证同一个事务中多次读取的数据是一致的，这是 MySQL InnoDB 引擎的默认隔离级别，
串行化（Serializable），并发事务之间是串行化的，通常意味着读取需要获取共享读锁，更新需要获取排他写锁，如果 SQL 使用 WHERE 语句，还会获取区间锁（MySQL 以 GAP 锁形式实现，可重复读级别中默认也会使用），这是最高的隔离级别。
MySQL 默认的事务隔离级别为 repeatable-read

MySQL 的复制原理以及流程

基本原理流程，3 个线程以及之间的关联；

```
主：binlog 线程—记录下所有改变了数据库数据的语句，放进 master 上的 binlog 中；
从：io 线程—在使用 start slave 之后，负责从 master 上拉取 binlog 内容，放进 自己的 relay log 中；
从：sql 执行线程—执行 relay log 中的语句；
```

MySQL 数据库 cpu 飙升到 500%的话他怎么处理？

1、列出所有进程 show processlist,观察所有进程 ,多秒没有状态变化的(干掉)
2、查看超时日志或者错误日志 (做了几年开发,一般会是查询以及大批量的插入会导致 cpu 与 i/o 上涨,当然不排除网络状态突然断了,,导致一个请求服务器只接受到一半，比如 where 子句或分页子句没有发送,,当然的一次被坑经历)

sql 优化各种方法

(1)、explain 出来的各种 item 的意义；

```
select_type:
    表示查询中每个 select 子句的类型 type:
    表示 MySQL 在表中找到所需行的方式，又称“访问类型”possible_keys:
    指出 MySQL 能使用哪个索引在表中找到行，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用 key:
    显示 MySQL 在查询中实际使用的索引，若没有使用索引，显示为 NULLkey_len:
    表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度
ref
    表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值
Extra
    包含不适合在其他列中显示但十分重要的额外信息
```

(2)、profile 的意义以及使用场景；

```
查询到 SQL 会执行多少时间，并看出 CPU/Memory 使用量，执行过程中 Systemlock，Table lock 花多少时间等等
```

你是如何监控你们的数据库的？你们的慢日志都是怎么查询的？

```
监控的工具有很多，例如 zabbix，lepus，我这里用的是 lepus
```

你是否做过主从一致性校验，如果有，怎么做的，如果没有，你打算怎么做？

主从一致性校验有多种工具 例如 checksum、mysqldiff、pt-table-checksum 等

你们数据库是否支持 emoji 表情，如果不支持，如何操作？

如果是 utf8 字符集的话，需要升级至 utf8_mb4 方可支持

开放性问题：据说是腾讯的

一个 6 亿的表 a，一个 3 亿的表 b，通过外间 tid 关联，你如何最快的查询出满足条件的第 50000 到第 50200 中的这 200 条数据记录。

```
1、如果 A 表 TID 是自增长,并且是连续的,B 表的 ID 为索引 select * from a,b where a.tid = b.id and a.tid>500000 limit 200;
2、如果 A 表的 TID 不是连续的,那么就需要使用覆盖索引.TID 要么是主键,要么是辅助索引,B 表 ID 也需要有索引。select * from b , (select tid from a limit 50000,200) a where b.id = a .tid;
```

数据库的乐观锁和悲观锁是什么？

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。
悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作
乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性

如何通俗地理解三个范式？

第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；
第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；
第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。。
范式化设计优缺点:

优点:可以尽量得减少数据冗余，使得更新快，体积小缺点:对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引优化
反范式化:

优点:可以减少表得关联，可以更好得进行索引优化缺点:数据冗余以及数据异常，数据得修改需要更多的成本

MySQL 的 binlog 有有几种录入格式？分别有什么区别？

有三种格式，statement，row 和 mixed。

statement 模式下，每一条会修改数据的 sql 都会记录在 binlog 中。不需要记录每一行的变化，减少了 binlog 日志量，节约了 IO，提高性能。由于 sql 的执行是有上下文的，因此在保存的时候需要保存相关的信息，同时还有一些使用了函数之类的语句无法被记录复制。

row 级别下，不记录 sql 语句上下文相关信息，仅保存哪条记录被修改。记录单元为每一行的改动，基本是可以全部记下来但是由于很多操作，会导致大量行的改动(比如 alter table)，因此这种模式的文件保存的信息太多，日志量太大。

mixed，一种折中的方案，普通操作使用 statement 记录，当无法使用 statement 的时候使用 row。

此外，新版的 MySQL 中对 row 级别也做了一些优化，当表结构发生变化的时候，会记录语句而不是逐行记录。

这些的话是在下面的直播课中讲解过很多期，并且在不断更新中
是结合企业的一些应用场景讲解的，能够帮助学员突破思维或者带着实战，来帮助大家掌握学习的

2020 年 10 月 8 日 晚上 20-23 点 腾讯课堂 大型直播课

php 高并发解决方案专场-亿级 PV 高并发架构案例深度剥析

上课时间：20:00

授课讲师：PHP 学院院长-peter 老师

课程链接：https://ke.qq.com/course/328509