高级 Web 开发求职必备 MySQL 面试知识点

前言

今互联网行业用的最多就是 MySQL，然而对于高级 Web 面试者，尤其对于寻找 30k 下工作的求职者，很多 MySQL 相关知识点基本都会涉及，并且由于是讲解面试题，可能会比较由浅入深，问答互动的方式，穿插讲解知识点。所以我尽量详细透彻讲解涉及的知识点，希望对大家有点帮助哈。废话不多说，开启问答面试之旅。

1、MySQL 相关存储引擎

因为目前大多互联网公司，用的最多的存储引擎就是 myisam 和 innodb。大多问 MySQL 有哪些存储引擎啊？都有什么区别？请详细说明一下，基本都是想面试者回答关于 myisam 和 innodb 引擎。

1.1）、面试题分析

mysql 支持的存储引擎有很多种，innodb、myisam、memory 等，但是面试官可能想问就 2 种，其实经常用的就一种，innodb，myisam 以前可能还有一些场景会用，现在用的已经非常少了，尤其在 oltp 业务中，一旦涉及事务、高并发等场景。

1.1.1）、myisam

myisam：不支持事务，不支持外键约束，索引文件和数据文件分开，这样在内存里可以缓存更多的索引，对查询的性能会更好，适用于那种少量的插入，大量查询的场景。

最经典的就是报表系统，比如大数据的报表系统，常见的就是走 hadoop 生态来搞，hdfs 来存储数据，然后基于 hive 来进行数仓建模，每次 hive 跑出来的数据都用 sqoop 从 hive 中导出到 mysql 中去。然后基于 mysql 的在线查询，就接上 php 写个简单的 web 系统，每个报表开发一套代码，写 sql 查数据，组织数据，按照前端要求的格式返回数据，展现出来一个报表。

这种报表系统，是最适合 mysql 的 myisam 存储引擎的，不需要事务，就是一次性批量导入，接下来一天之内就是纯查询了。

1.1.2）、innodb

现在一般用 mysql 都是 innodb，很少用其他的存储引擎，而且国内用其他存储引擎的场景和公司也不多，所以用 innodb 就可以了，而且这个也是 mysql 5.5 之后的默认官方存储引擎。

主要特点就是支持事务，走聚簇索引，强制有主键，支持外键约束，并且高并发、大数据量、高可用等有相关成熟的数据库架构，分库分表、读写分离、主备切换，全部都可以基于 innodb 存储引擎的，稍微要求高点的面试官，可能会问你 innodb 存储引擎怎么玩支撑大数据量和高并发，如何用读写分离支撑高可用和高可用读，如何分摊 db 压力，支撑更高 OPS，基本就是涉及更深入的 innodb 原理的东西，可以建议看下一本书：innodb 技术内幕。

1.2）、总结

基本关于 mysql 存储引擎，可能开始面试官只是想问 myisam 和 innodb 简单的事务、数据存储原理等区别，再深入问可能就想知道 innodb 在高并发、大数据量、高可用架构下的优势。

关于本题，只是大致的抛砖引玉，要是去讲解高并发、大数据量等，那涉及的知识太多了哈。

2、MySQL 索引实现原理

MySQ 索引的原理和数据结构能介绍一下吗、MySQL 聚簇索引和非聚簇索引的区别是什么、他们分别是如何存储的？使用 MySQL 索引都有哪些原则、MySQL 复合索引如何使用。

基本会涉及这些知识点，稍微好点公司的面试官哈。

2.1）、索引存储结构

其实大多面试官问你 mysql 的索引底层是什么数据结构实现的，可能还会现场让你优化 sql，为什么如此优化呢；还可能问你数据库常见的使用规则呢。

mysql 的索引就是用一个数据结构组织某一列的数据，然后如果你要根据那一列的数据查询的时候，就可以不用全表扫描，只要根据那个特定的数据结构去找到那一列的值，然后找到对应的行的物理地址即可。

mysql 的索引实现是基于 B+ 树，这样查找数据的高度就算数据量很大，也很低，关于 B+ 树原理和实现，可以参考任何一本数据结构的书籍了解下哈。

myisam 存储引擎的索引中，每个叶子节点的 data 存放的是数据行的物理地址，比如 0x07 之类的东西，一行一行的，每行对应一个物理地址。

innodb 存储引擎的索引实现，跟 myisam 最大的区别在于，innodb 的数据文件本身就是个索引文件，就是 key 就是主键，然后叶子节点的 data 就是那个数据行。

innodb 存储引擎，必须有主键，可以默认内置的就会根据主键（6 字节的 rowid）建立一个索引，叫做聚簇索引，innodb 的数据文件本身同时也是个索引文件，这个索引就是默认根据主键建立的叫做聚簇索引。

innodb 这种原生的数据文件就是索引文件的组织结构，就叫默认的主键索引为聚簇索引。就是因为这个原因，innodb 表是要求必须有主键的，但是 myisam 表不要求必须有主键。另外一个是，innodb 存储引擎下，如果对某个非主键的字段创建个索引，那么最后那个叶子节点的值就是主键的值，因为可以用主键的值到聚簇索引里根据主键值再次查找到数据。

一般 innodb 表里，建议统一用 `auto_increment` 自增值作为主键值，因为这样可以保持聚簇索引直接加记录就可以，如果用那种不是单调递增的主键值，可能会导致 b+ 树分裂后重新组织，会浪费时间。这也就是为啥 innodb 下不要用 UUID 生成的超长字符串作为主键？因为这么玩儿会导致所有的索引的 data 都是那个主键值，最终导致索引会变得过大，浪费很多磁盘空间。

2.2）、索引使用规则

尽量根据 where 条件、选择列，综合考虑建立最合适的索引。

2.2.1）、全列匹配

尽量 where 条件里，根据最左匹配原则，建立联合索引。

比如 `select name,place_id from car_parking_place where car_id=xxxx and ctime='2018-01-10 00: 00:00';` 就可以建立 `create index (car_id,ctime)`。

2.2.2)、最左前缀匹配

如果你的 sql 里，正好就用到了联合索引最左边的一个或者几个列表，那么也可以用上这个索引，在索引里查找的时候就用最左边的几个列。

2.2.3）、前缀匹配

如果你不是等值的，比如 `=`， `>=`， `<=` 的操作，是 like 操作，那么必须要是 `like 'XX%'` 这种才可以用上索引。

2.2.4）、范围列匹配

如果你是范围查询，比如 `>=`， `<=`， `between` 操作，你只能是符合最左前缀的规则才可以范围，范围之后的列就不用索引。

2.2.5）、禁用函数

目前 mysql 还不支持函数索引，如果你对某个列用了函数，比如 substring 之类的东西，那么那一列不用索引。

2.3）、索引的缺点

索引是有缺点的，比如常见的就是会增加磁盘消耗，因为要占用磁盘文件，同时高并发的时候频繁插入和修改索引，会导致性能损耗的。尽量创建少的索引，比如说一个表一两个索引，两三个索引，十几个索引，高并发场景下还可以。

尽量对表中唯一性相对较高、经常查询的字段建立索引，那种比如 sex（性别），世上就男女可能2种值，建立索引，几乎跟全表扫描没什么区别。

3、MySQL 数据库锁

面试官一般聊锁，就是问 myisam 和 innodb 存储引擎下的表锁、行锁，还有可能想知道死锁咋办，如何检测业务中锁。

3.1）、mysql 锁

一般 myisam 会加表锁，就是 myisam 引擎下，执行查询的时候，会默认加个表共享锁，也就是表读锁，这个时候别人只能来查，不能写数据的；然后 myisam 写的时候，也会加个表独占锁，也就是表写锁，别人不能读也不能写。

这个 myisam 因为很少用了，面试的时候大致说下就可以啦。其实很多人也发现了，myisam 其实在实际生产中，也就新闻资讯、报表系统等，也就是纯粹读写比，读超级多的系统在用了，压根不涉及事务。可以简单理解为 olap 业务。

innodb 的行锁有共享锁（S）和排他锁（X），两种，其实说白了呢，共享锁就是，多个事务都可以加共享锁读同一行数据，但是别的事务不能写这行数据；排他锁，就是就一个事务可以写这行数据，别的事务只能读，不能写。

innodb 的表锁，分成意向共享锁，就是说加共享行锁的时候，必须先加这个共享表锁；还有一个意向排他锁，就是说，给某行加排他锁的时候，必须先给表加排他锁。这个表锁，是 innodb 引擎自动加的，不用你自己去加。

insert、update、delete，innodb 会自动给那一行加行级排他锁。

select，innodb 啥锁都不加，因为 innodb 默认实现了可重复读，也就是 mvcc 机制，所以多个事务随便读一个数据，一般不会有冲突，大家就读自己那个快照就可以了，不涉及到什么锁的问题。但是 innodb 从来不会自己主动加个共享锁的，除非你用下面的语句自己手动加个锁：

- 手动加共享锁：`select * fro m table where id=xxx lock in share mode`，那你就给那一行加了个共享锁，其他事务就不能来修改这行数据了.

- 手动加排他锁：`select * from table where id=xxx for update`，那你就给那一行加了个排他锁，意思就是你准备修改，别的事务就别修改了，别的事务的修改会卡住。这个要慎用，一般线上系统禁止使用。

3.2）、死锁场景

一个 client 下：

```
session a：select * from car where id=1 for update
session b:select * from car where id=2 for update
```

另外一个 client 下：

```
session a：select * from car where id=2 for update
session b:select * from car where id=1 for update
```

如此上述场景就会导致死锁，可以根据 `innodb_lock_waits`、`innodb_locks` 等查询具体是那些 sql 导致，具体分析下。

4、MySQL 的 SQL 调优

MySQL 里最直接的优化就是保证减少 io 请求，尽量 90% 多业务走单表扫描，需要计算或者关联的业务，尽量放在程序层完成。然而此时就会有疑问，都单表了，还有什么好优化的呢？

如果一个线上业务 SQL 比较慢，十有八九就是因为那个 SQL 没有用索引，所以这个时候，第一步就是去看 MySQL 的执行计划，看看那个 SQL 有没有用到索引，如果没有，那么就改写一下 SQL 让他用上索引，或者是额外加个索引。

关于 mysql 执行计划，可以用 explain 或 explain extended 分析下执行计划，重点关注下 key（用到那个索引）、rows（扫描行数）、`extra：using filesort`（需要额外进行排序），`using temporary`（mysql 构建了临时表，比如排序的时候），`using where`（就是对索引扫出来的数据再次根据 where 来过滤出了结果）重点的信息，基本也就能定位 sql 性能的问题了。

5、MySQL 分表分库全局 ID

mysql 分库分表确实能解决不少问题，也能让数据库支撑更大的并发、大数据量业务，可是分库分表后必然面对的一个问题就是 id 咋生成？id 生成后如何保持全局唯一性。

5.1）、数据库自增id

系统里每次得到一个 id，都是往一个库的一个表里插入一条没什么业务含义的数据，然后获取一个数据库自增的一个 id。拿到这个 id 之后再往对应的分库分表里去写入。这样确实可以解决，但是在高并发业务场景下，就会发生瓶颈，瞬间那一刻会出现脏数据。所以这个办法只能适合非高并发下的分表分库，如何生成全局id。

5.2）、uuid

本地生成，不要基于数据库来了；不好之处就是，uuid 太长了，作为主键性能太差了，不适合用于主键。

如果你是要随机生成个什么文件名了，编号之类的，你可以用 uuid，但是作为主键是不能用 uuid 的。

5.3）、获取当前系统时间

这个就是获取当前时间即可，但是问题是，并发很高的时候，比如一秒并发几千，会有重复的情况，这个是肯定不合适的。

一般如果用这个方案，是将当前时间跟很多其他的业务字段拼接起来，作为一个 id，如果业务上你觉得可以接受，那么也是可以的。你可以将别的业务字段值跟当前时间拼接起来，组成一个全局唯一的标识，订单号，时间戳 + 用户 id + 业务唯一标识。

5.4）、snowflake 算法*

在系统小时，唯一标识的产生，可以利用公用模块来处理，比如数据库表的唯一键、或者缓存的唯一 id 等等方式。但在分布式高并发的系统中，如果还是这样使用公共模块，就会产生很大的风险和瓶颈。

snowflake 是 Twitter 开源的分布式 ID 生成算法，结果是一个 long 型的 ID。其核心思想是：使用 41 bit 作为毫秒数，10 bit 作为机器的 ID（5 个 bit 是数据中心，5 个 bit 的机器 ID），12bit 作为毫秒内的流水号（意味着每个节点在每毫秒可以产生 4096 个 ID），最后还有一个符号位，永远是 0。

这个算法单机每秒内理论上最多可以生成 1000 * (2 ^ 12)，也就是 409.6 万个 ID，（吼吼，这个得了的快啊）。

相关 Java 实现代码：

https://www.cnblogs.com/relucent/p/4955340.html

综上所述，也基本是笔者自己去面试大厂，或者自己作为面试官招聘高工、资深时都会问的问题，上述比如全局 id 很多人都没怎么去了解过、mysql 锁很多人更没那意识等，希望通过一些问题能让读者可以学习到一点知识。

因为内容有点总结的感觉，而不是详细阐述，毕竟面试时，不可能让你一个问题跟背书一样吧，可能某些问题讲的不够细，不深入，不过基本大体阐述了问题的核心点啦，如有疑问，可以留言交流，非常谢谢！