阿里面试中关于索引有关的问题以及知识点

索引概念、索引模型

我们是怎么聊到索引的呢，是因为我提到我们的业务量比较大，每天大概有几百万的新数据生成，于是有了以下对话：

面试官：你们每天这么大的数据量，都是保存在关系型数据库中吗？

我：是的，我们线上使用的是 MySQL 数据库

面试官：每天几百万数据，一个月就是几千万了，那你们有没有对于查询做一些优化呢？

我：我们在数据库中创建了一些索引（我现在非常后悔我当时说了这句话）。

这里可以看到，阿里的面试官并不会像有一些公司一样拿着题库一道一道的问，而是会根据面试者做过的事情以及面试过程中的一些内容进行展开。

面试官：那你能说说什么是索引吗？

我：（这道题肯定难不住我啊）索引其实是一种数据结构，能够帮助我们快速的检索数据库中的数据。

面试官：那么索引具体采用的哪种数据结构呢？

我：（这道题我也背过）常见的 MySQL 主要有两种结构：Hash 索引和 B+ Tree 索引，我们使用的是 InnoDB 引擎，默认的是 B+树。

这里我耍了一个小心机，特意说了一下索引和存储引擎有关。希望面试官可以问我一些关于存储引擎的问题。

面试官：既然你提到 InnoDB 使用的 B+ Tree 的索引模型，那么你知道为什么采用 B+ 树吗？这和 Hash 索引比较起来有什么优缺点吗？

我：（突然觉得这道题有点难，但是我还是凭借着自己的知识储备简单的回答上一些）因为 Hash 索引底层是哈希表，哈希表是一种以 key-value 存储数据的结构，所以多个数据在存储关系上是完全没有任何顺序关系的，所以，对于区间查询是无法直接通过索引查询的，就需要全表扫描。所以，哈希索引只适用于等值查询的场景。而 B+ Tree 是一种多路平衡查询树，所以他的节点是天然有序的（左子节点小于父节点、父节点小于右子节点），所以对于范围查询的时候不需要做全表扫描。

面试官：除了上面这个范围查询的，你还能说出其他的一些区别吗？

我：（这个题我回答的不好，事后百度了一下）

科普时间：B+ Tree 索引和 Hash 索引区别 哈希索引适合等值查询，但是不无法进行范围查询 哈希索引没办法利用索引完成排序 哈希索引不支持多列联合索引的最左匹配规则 如果有大量重复键值得情况下，哈希索引的效率会很低，因为存在哈希碰撞问题

聚簇索引、覆盖索引

面试官：刚刚我们聊到 B+ Tree ，那你知道 B+ Tree 的叶子节点都可以存哪些东西吗？

我：InnoDB 的 B+ Tree 可能存储的是整行数据，也有可能是主键的值。

面试官：那这两者有什么区别吗？

我：（当他问我叶子节点的时候，其实我就猜到他可能要问我聚簇索引和非聚簇索引了）在 InnoDB 里，索引 B+ Tree 的叶子节点存储了整行数据的是主键索引，也被称之为聚簇索引。而索引 B+ Tree 的叶子节点存储了主键的值的是非主键索引，也被称之为非聚簇索引。

面试官：那么，聚簇索引和非聚簇索引，在查询数据的时候有区别吗？

我：聚簇索引查询会更快？

面试官：为什么呢？

我：因为主键索引树的叶子节点直接就是我们要查询的整行数据了。而非主键索引的叶子节点是主键的值，查到主键的值以后，还需要再通过主键的值再进行一次查询。

面试官：刚刚你提到主键索引查询只会查一次，而非主键索引需要回表查询多次。（后来我才知道，原来这个过程叫做回表）是所有情况都是这样的吗？非主键索引一定会查询多次吗？

我：（额、这个问题我回答的不好，后来我自己查资料才知道，通过覆盖索引也可以只查询一次）

科普时间——覆盖索引 覆盖索引（covering index）指一个查询语句的执行只用从索引中就能够取得，不必从数据表中读取。也可以称之为实现了索引覆盖。 当一条查询语句符合覆盖索引条件时，MySQL 只需要通过索引就可以返回查询所需要的数据，这样避免了查

到索引后再返回表操作，减少 I/O 提高效率。 如，表 covering_index_sample 中有一个普通索引 idx_key1_key2(key1,key2)。当我们通过 SQL 语句：select key2 from covering_index_sample where key1 = 'keytest';的时候，就可以通过覆盖索引查询，无需回表。

联合索引、最左前缀匹配

面试官：不知道的话没关系，想问一下，你们在创建索引的时候都会考虑哪些因素呢？

我：我们一般对于查询概率比较高，经常作为 where 条件的字段设置索引

面试官：那你们有用过联合索引吗？

我：用过呀，我们有对一些表中创建过联合索引。

面试官：那你们在创建联合索引的时候，需要做联合索引多个字段之间顺序你们是如何选择的呢？

我：我们把识别度最高的字段放到最前面。

面试官：为什么这么做呢？

我：（这个问题有点把我问蒙了，稍微有些慌乱）这样的话可能命中率会高一点吧。。。

面试官：那你知道最左前缀匹配吗？

我：（我突然想起来原来面试官是想问这个，怪自己刚刚为什么就没想到这个呢。）哦哦哦。您刚刚问的是这个意思啊，在创建多列索引时，我们根据业务需求，where 子句中使用最频繁的一列放在最左边，因为 MySQL 索引查询会遵循最左前缀匹配的原则，即最左优先，在检索数据时从联合索引的最左边开始匹配。所以当我们创建一个联合索引的时候，如(key1,key2,key3)，相当于创建了（key1）、(key1,key2)和(key1,key2,key3)三个索引，这就是最左匹配原则。

虽然我一开始有点懵，没有联想到最左前缀匹配，但是面试官还是引导了我。很友善。

索引下推、查询优化

面试官：你们线上用的 MySQL 是哪个版本啊呢？

我：我们 MySQL 是 5.7

面试官：那你知道在 MySQL 5.6 中，对索引做了哪些优化吗？

我：不好意思，这个我没有去了解过。（事后我查了一下，有一个比较重要的 ：Index Condition Pushdown Optimization）

科普时间—— Index Condition Pushdown（索引下推） MySQL 5.6 引入了索引下推优化，默认开启，使用 SET optimizer_switch = ‘index_condition_pushdown=off’；可以将其关闭。官方文档中给的例子和解释如下： people 表中（zipcode，lastname，firstname）构成一个索引

```
SELECT * FROM people WHERE zipcode='95054' AND lastname LIKE
'%etrunia%' AND address LIKE '%Main Street%';
```

如果没有使用索引下推技术，则 MySQL 会通过 zipcode='95054’ 从存储引擎中查询对应的数据，返回到 MySQL 服务端，然后 MySQL 服务端基于 lastname LIKE '%etrunia%'和 address LIKE '%Main Street%'来判断数据是否符合条件。 如果使用了索引下推技术，则 MYSQL 首先会返回符合 zipcode='95054’ 的索引，然后根据 lastname LIKE '%etrunia%'和 address LIKE '%Main Street%'来判断索引是否符合条件。如果符合条件，则根据该索引来定位对应的数据，如果不符合，则直接 reject 掉。 有了索引下推优化，可以在有 like 条件查询的情况下，减少回表次数。

面试官：你们创建的那么多索引，到底有没有生效，或者说你们的 SQL 语句有没有使用索引查询你们有统计过吗？

我：这个还没有统计过，除非遇到慢 SQL 的时候我们才会去排查

面试官：那排查的时候，有什么手段可以知道有没有走索引查询呢？

我：可以通过 explain 查看 sql 语句的执行计划，通过执行计划来分析索引使用情况

面试官：那什么情况下会发生明明创建了索引，但是执行的时候并没有通过索引呢？

我：（依稀记得和优化器有关，但是这个问题并没有回答好）

科普时间——查询优化器 一条 SQL 语句的查询，可以有不同的执行方案，至于最终选择哪种方案，需要通过优化器进行选择，选择执行成本最低的方案。 在一条单表查询语句真正执行之前，MySQL 的查询优化器会找出执行该语句所有可能使用的方案，对比之后找出成本最低的方案。这个成本最低的方案就是所谓的执行计划。 优化过程大致如下：

1、根据搜索条件，找出所有可能使用的索引 2、计算全表扫描的代价 3、计算使用不同索引执行查询的代价 4、对比各种执行方案的代价，找出成本最低的那一个

面试官：哦，索引有关的知识我们暂时就问这么多吧。你们线上数据的事务隔离级别是什么呀？

我：(后面关于事务隔离级别的问题了，就不展开了)

感觉是因为我回答的不够好，如果这几个索引问题我都会的话，他还会追问更多，恐怕会被虐的更惨

总结&感悟

以上，就是一次面试中关于索引部分知识的问题以及我整理的答案。感觉这次面试过程中关于索引的知识，自己大概能够回答的内容占 70%左右，但是自信完全答对的内容只占 50%左右，看来自己索引有关的知识了解的还是不够多。

通过这次面试，发现像阿里这种大厂对于底层知识还是比较看重的，我以前以为关于索引最多也就问一下 Hash 和 B+有什么区别，没想到最后都能问到查询优化器上面。

简单整理下事务隔离级别的知识如下，了解更多请查阅相关资料。

谈到事务最先想到的就是 ACID 属性（Atomicity 原子性、Consistency 一致性、Isolation 隔离性、Durability 持久性），今天主要介绍一下 MySQL 的隔离属性。

MySQL 的事务的隔离级别分为：未提交读(read uncommitted)、已提交读(read committed)、可重复读(repeatable read)、串行化(serializable)。

未提交读：一个事务可以读取到，另外一个事务尚未提交的变更。

已提交读：一个事务提交后，其变更才会被另一个事务读取到。

可重复读：在一个事务执行的过程中所读取到的数据，和事务启动时所看到的一致。

串行化：当操作一行数据时，读写分别都会加锁。当出现读写锁互斥时，会排队串行执行。