吊打面试官的 MySQL 灵魂 100 问

前言

本文主要受众为开发人员,所以不涉及到 MySQL 的服务部署等操作,且内容较多,大家准备好耐心和瓜子矿泉水.

前一阵系统的学习了一下 MySQL,也有一些实际操作经验,偶然看到一篇和 MySQL 相关的面试文章,发现其中的一些问题自己也回答不好,虽然知识点大部分都知道,但是无法将知识串联起来.

因此决定搞一个 MySQL 灵魂 100 问,试着用回答问题的方式,让自己对知识点的理解更加深入一点.

此文不会事无巨细的从 select 的用法开始讲解 mysql,主要针对的是开发人员需要知道的一些 MySQL 的知识点,主要包括索引,事务,优化等方面,以在面试中高频的问句形式给出答案.

索引相关

关于 MySQL 的索引,曾经进行过一次总结,文章链接在这里 Mysql 索引原理及其优化.

1. 什么是索引?

索引是一种数据结构,可以帮助我们快速的进行数据的查找.

2. 索引是个什么样的数据结构呢?

索引的数据结构和具体存储引擎的实现有关, 在 MySQL 中使用较多的索引有 Hash 索引,B+树索引等,而我们经常使用的 InnoDB 存储引擎的默认索引实现为:B+树索引.

3. Hash 索引和 B+树所有有什么区别或者说优劣呢?

首先要知道 Hash 索引和 B+树索引的底层实现原理:

hash 索引底层就是 hash 表,进行查找时,调用一次 hash 函数就可以获取到相应的键值,之后进行回表查询获得实际数据.B+树底层实现是多路平衡查找树.对于每一次的查询都是从根节点出发,查找到叶子节点方可以获得所查键值,然后根据查询判断是否需要回表查询数据.

那么可以看出他们有以下的不同:

- hash 索引进行等值查询更快(一般情况下),但是却无法进行范围查询.

因为在 hash 索引中经过 hash 函数建立索引之后,索引的顺序与原顺序无法保持一致,不能支持范围查询.而 B+树的的所有节点皆遵循(左节点小于父节点,右节点大于父节点,多叉树也类似),天然支持范围.

hash 索引不支持使用索引进行排序,原理同上.

hash 索引不支持模糊查询以及多列索引的最左前缀匹配.原理也是因为 hash 函数的不可预测.**AAAA** 和 **AAAAB** 的索引没有相关性.

hash 索引任何时候都避免不了回表查询数据,而 B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询.

hash 索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在大量重复的时候,发生 hash 碰撞,此时效率可能极差.而 B+树的查询效率比较稳定,对于所有的查询都是从根节点到叶子节点,且树的高度较低.

因此,在大多数情况下,直接选择 B+树索引可以获得稳定且较好的查询速度.而不需要使用 hash 索引.

4. 上面提到了 B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引?

在 B+树的索引中,叶子节点可能存储了当前的 key 值,也可能存储了当前的 key 值以及整行的数据,这就是聚簇索引和非聚簇索引. 在 InnoDB 中,只有主键索引是聚簇索引,如果没有主键,则挑选一个唯一键建立聚簇索引.如果没有唯一键,则隐式的生成一个键来建立聚簇索引.

当查询使用聚簇索引时,在对应的叶子节点,可以获取到整行数据,因此不用再次进行回表查询.

5. 非聚簇索引一定会回表查询吗?

不一定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进行回表查询.

举个简单的例子,假设我们在员工表的年龄上建立了索引,那么当进行 `select age from employee where age < 20` 的查询时,在索引的叶子节点上,已经包含了 age 信息,不会再次进行回表查询.

6. 在建立索引的时候,都有哪些需要考虑的因素呢?

建立索引的时候一般要考虑到字段的使用频率,经常作为条件进行查询的字段比较适合.如果需要建立联合索引的话,还需要考虑联合索引中的顺序.此外也要考虑其他方面,比如防止过多的所有对表造成太大的压力.这些都和实际的表结构以及查询方式有关.

7. 联合索引是什么?为什么需要注意联合索引中的顺序?

MySQL 可以使用多个字段同时建立一个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建立索引时的字段顺序挨个使用,否则无法命中索引.

具体原因为:

MySQL 使用索引时需要索引有序,假设现在建立了"name,age,school"的联合索引,那么索引的排序为: 先按照 name 排序,如果 name 相同,则按照 age 排序,如果 age 的值也相等,则按照 school 进行排序.

当进行查询时,此时索引仅仅按照 name 严格有序,因此必须首先使用 name 字段进行等值查询,之后对于匹配到的列而言,其按照 age 字段严格有序,此时可以使用 age 字段用做索引查找,,,以此类推.因此在建立联合索引的时候应该注意索引列的顺序,一般情况下,将查询需求频繁或者字段选择性高的列放在前面.此外可以根据特例的查询或者表结构进行单独的调整.

8. 创建的索引有没有被使用到?或者说怎么才可以知道这条语句运行很慢的原因?

MySQL 提供了 explain 命令来查看语句的执行计划,MySQL 在执行某个语句之前,会将该语句过一遍查询优化器,之后会拿到对语句的分析,也就是执行计划,其中包含了许多信息. 可以通过其中和索引有关的信息来分析是否命中了索引,例如 possilbe_key,key,key_len 等字段,分别说明了此语句可能会使用的索引,实际使用的索引以及使用的索引长度.

9. 那么在哪些情况下会发生针对该列创建了索引但是在查询的时候并没有使用呢?

使用不等于查询,

列参与了数学运算或者函数

在字符串 like 时左边是通配符.类似于'%aaa'.

当 mysql 分析全表扫描比使用索引快的时候不使用索引.

当使用联合索引,前面一个条件为范围查询,后面的即使符合最左前缀原则,也无法使用索引.

以上情况,MySQL 无法使用索引.

事务相关

1. 什么是事务?

理解什么是事务最经典的就是转账的栗子,相信大家也都了解,这里就不再说一边了.

事务是一系列的操作,他们要符合 ACID 特性.最常见的理解就是:事务中的操作要么全部成功,要么全部失败.但是只是这样还不够的.

2. ACID 是什么?可以详细说一下吗?

A=Atomicity

原子性,就是上面说的,要么全部成功,要么全部失败.不可能只执行一部分操作.

C=Consistency

系统(数据库)总是从一个一致性的状态转移到另一个一致性的状态,不会存在中间状态.

I=Isolation

隔离性: 通常来说:一个事务在完全提交之前,对其他事务是不可见的.注意前面的通常来说加了红色,意味着有例外情况.

D=Durability

持久性,一旦事务提交,那么就永远是这样子了,哪怕系统崩溃也不会影响到这个事务的结果.

3. 同时有多个事务在进行会怎么样呢?

多事务的并发进行一般会造成以下几个问题:

脏读: A 事务读取到了 B 事务未提交的内容,而 B 事务后面进行了回滚.

不可重复读: 当设置 A 事务只能读取 B 事务已经提交的部分,会造成在 A 事务内的两次查询,结果竟然不一样,因为在此期间 B 事务进行了提交操作.

幻读: A 事务读取了一个范围的内容,而同时 B 事务在此期间插入了一条数据.造成"幻觉".

4. 怎么解决这些问题呢?MySQL 的事务隔离级别了解吗?

MySQL 的四种隔离级别如下:

- 未提交读(READ UNCOMMITTED)

这就是上面所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,而之后该事务进行了回滚).

这个级别的性能没有足够大的优势,但是又有很多的问题,因此很少使用.

- 已提交读(READ COMMITTED)

其他事务只能读取到本事务已经提交的部分.这个隔离级别有 不可重复读的问题,在同一个事务内的两次读取,拿到的结果竟然不一样,因为另外一个事务对数据进行了修改.

- REPEATABLE READ(可重复读)

可重复读隔离级别解决了上面不可重复读的问题(看名字也知道),但是仍然有一个新问题,就是 幻读,当你读取 id> 10 的数据行时,对涉及到的所有行加上了读锁,此时例外一个事务新插入了一条 id=11 的数据,因为是新插入的,所以不会触发上面的锁的排斥,那么进行本事务进行下一次的查询时会发现有一条 id=11 的数据,而上次的查询操作并没有获取到,再进行插入就会有主键冲突的问题.

- SERIALIZABLE(可串行化)

这是最高的隔离级别,可以解决上面提到的所有问题,因为他强制将所以的操作串行执行,这会导致并发性能极速下降,因此也不是很常用.

5. Innodb 使用的是哪种隔离级别呢?

InnoDB 默认使用的是可重复读隔离级别.

6. 对 MySQL 的锁了解吗?

当数据库有并发事务的时候,可能会产生数据的不一致,这时候需要一些机制来保证访问的次序,锁机制就是这样的一个机制.

就像酒店的房间,如果大家随意进出,就会出现多人抢夺同一个房间的情况,而在房间上装上锁,申请到钥匙的人才可以入住并且将房间锁起来,其他人只有等他使用完毕才可以再次使用.

7. MySQL 都有哪些锁呢?像上面那样子进行锁定岂不是有点阻碍并发效率了?

从锁的类别上来讲,有共享锁和排他锁.

共享锁: 又叫做读锁. 当用户要进行数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.

排他锁: 又叫做写锁. 当用户要进行数据的写入时,对数据加上排他锁.排他锁只可以加一个,他和其他的排他锁,共享锁都相斥.

用上面的例子来说就是用户的行为有两种,一种是来看房,多个用户一起看房是可以接受的. 一种是真正的入住一晚,在这期间,无论是想入住的还是想看房的都不可以.

锁的粒度取决于具体的存储引擎,InnoDB 实现了行级锁,页级锁,表级锁.

他们的加锁开销从大大小,并发能力也是从大到小.

表结构设计

1. 为什么要尽量设定一个主键?

主键是数据库确保数据行在整张表唯一性的保障,即使业务上本张表没有主键,也建议添加一个自增长的 ID 列作为主键.设定了主键之后,在后续的删改查的时候可能更加快速以及确保操作数据范围安全.

2. 主键使用自增 ID 还是 UUID?

推荐使用自增 ID,不要使用 UUID.

因为在 InnoDB 存储引擎中,主键索引是作为聚簇索引存在的,也就是说,主键索引的 B+树叶子节点上存储了主键索引以及全部的数据(按照顺序),如果主键索引是自增 ID,那么只需要不断向后排列即可,如果是 UUID,由于到来的 ID 与原来的大小不确定,会造成非常多的数据插入,数据移动,然后导致产生很多的内存碎片,进而造成插入性能的下降.

总之,在数据量大一些的情况下,用自增主键性能会好一些.

图片来源于《高性能 MySQL》: 其中默认后缀为使用自增 ID,_uuid 为使用 UUID 为主键的测试,测试了插入 100w 行和 300w 行的性能.

表 5-1：向InnoDB表插入数据的测试结果

表名	行数	时间（秒）	索引大小（MB）
userinfo	1 000 000	137	342
userinfo_uuid	1 000 000	180	544
userinfo	3 000 000	1233	1036
userinfo_uuid	3 000 000	4525	1707

关于主键是聚簇索引,如果没有主键,InnoDB 会选择一个唯一键来作为聚簇索引,如果没有唯一键,会生成一个隐式的主键.

> If you define a PRIMARY KEY on your table, InnoDB uses it as the clustered index.
>
> If you do not define a PRIMARY KEY for your table, MySQL picks the first UNIQUE index that has only NOT NULL columns as the primary key and InnoDB uses it as the clustered index.

3. 字段为什么要求定义为 not null?

MySQL 官网这样介绍:

> NULL columns require additional space in the rowto record whether their values are NULL. For MyISAM tables, each NULL columntakes one bit extra, rounded up to the nearest byte.

null 值会占用更多的字节,且会在程序中造成很多与预期不符的情况.

4. 如果要存储用户的密码散列,应该使用什么字段进行存储?

密码散列,盐,用户身份证号等固定长度的字符串应该使用 char 而不是 varchar 来存储,这样可以节省空间且提高检索效率.

存储引擎相关

1. MySQL 支持哪些存储引擎?

MySQL 支持多种存储引擎,比如 InnoDB,MyISAM,Memory,Archive 等等.在大多数的情况下,直接选择使用 InnoDB 引擎都是最合适的,InnoDB 也是 MySQL 的默认存储引擎.

1. InnoDB 和 MyISAM 有什么区别?

   InnoDB 支持事物，而 MyISAM 不支持事物

   InnoDB 支持行级锁，而 MyISAM 支持表级锁

   InnoDB 支持 MVCC，而 MyISAM 不支持

   InnoDB 支持外键，而 MyISAM 不支持

   InnoDB 不支持全文索引，而 MyISAM 支持。

零散问题

1. MySQL 中的 varchar 和 char 有什么区别.

char 是一个定长字段,假如申请了 `char(10)`的空间,那么无论实际存储多少内容.该字段都占用 10 个字符,而 varchar 是变长的,也就是说申请的只是最大长度,占用的空间为实际字符长度+1,最后一个字符存储使用了多长的空间.

在检索效率上来讲,char > varchar,因此在使用中,如果确定某个字段的值的长度,可以使用 char,否则应该尽量使用 varchar.例如存储用户 MD5 加密后的密码,则应该使用 char.

2. varchar(10)和 int(10)代表什么含义?

varchar 的 10 代表了申请的空间长度,也是可以存储的数据的最大长度,而 int 的 10 只是代表了展示的长度,不足 10 位以 0 填充.也就是说,int(1)和 int(10)所能存储的数字大小以及占用的空间都是相同的,只是在展示时按照长度展示.

3. 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,记录单元为语句.即每一个 sql 造成的影响会记录.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,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.

4. 超大分页怎么处理?

超大的分页一般从两个方向上来解决.

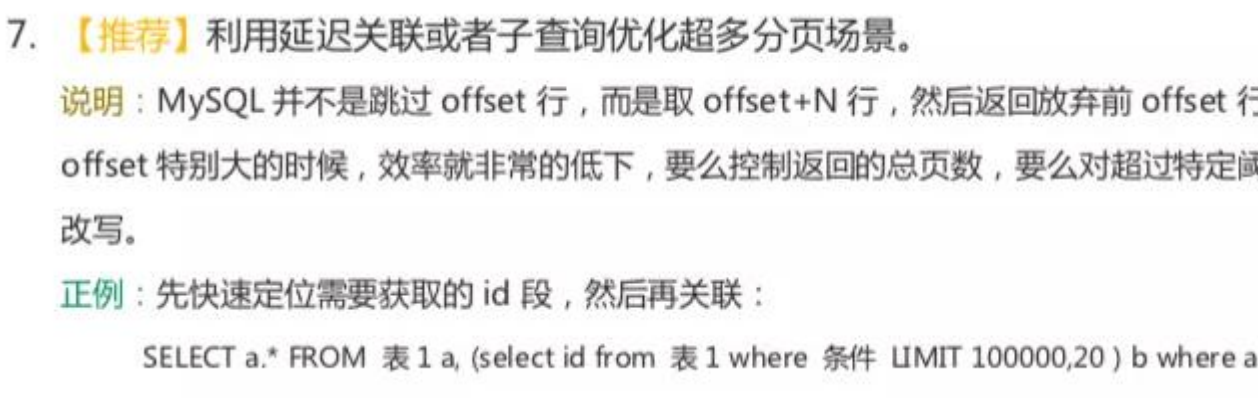
> 数据库层面,这也是我们主要集中关注的(虽然收效没那么大),类似于 `select * from table where age > 20 limit 1000000,10` 这种查询其实也是有可以优化的余地的. 这条语句需要 load1000000 数据然后基本上全部丢弃,只取 10 条当然比较慢. 当时我们可以修改为 `select * from table where id in (select id from table where age > 20 limit 1000000,10)`.这样虽然也 load 了一百万的数据,但是由于

索引覆盖,要查询的所有字段都在索引中,所以速度会很快. 同时如果 ID 连续的好,我们还可以 `select * from table where id > 1000000 limit 10`,效率也是不错的,优化的可能性有许多种,但是核心思想都一样,就是减少 load 的数据.

从需求的角度减少这种请求....主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止 ID 泄漏且连续被人恶意攻击.

解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至 redis 等 k-V 数据库中,直接返回即可.

在阿里巴巴《Java 开发手册》中,对超大分页的解决办法是类似于上面提到的第一种.


7. 【推荐】利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行

offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈

改写。

正例：先快速定位需要获取的 id 段，然后再关联：

SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20) b where a

5. 关心过业务系统里面的 sql 耗时吗?统计过慢查询吗?对慢查询都怎么优化过?

在业务系统中,除了使用主键进行的查询,其他的我都会在测试库上测试其耗时,慢查询的统计主要由运维在做,会定期将业务中的慢查询反馈给我们.

慢查询的优化首先要搞明白慢的原因是什么? 是查询条件没有命中索引?是 load 了不需要的数据列?还是数据量太大?

所以优化也是针对这三个方向来的,

首先分析语句,看看是否 load 了额外的数据,可能是查询了多余的行并且抛弃掉了,可能是加载了许多结果中并不需要的列,对语句进行分析以及重写.

分析语句的执行计划,然后获得其使用索引的情况,之后修改语句或者修改索引,使得语句可以尽可能的命中索引.

如果对语句的优化已经无法进行,可以考虑表中的数据量是否太大,如果是的话可以进行横向或者纵向的分表.

6. 上面提到横向分表和纵向分表,可以分别举一个适合他们的例子吗?

横向分表是按行分表.假设我们有一张用户表,主键是自增 ID 且同时是用户的 ID.数据量较大,有 1 亿多条,那么此时放在一张表里的查询效果就不太理想.我们可以根据主键 ID 进行分表,无论是按尾号分,或者按 ID 的区间分都是可以的. 假设按照尾号 0-99 分为 100 个表,那么每张表中的数据就仅有 100w.这时的查询效率无疑是可以满足要求的.

纵向分表是按列分表.假设我们现在有一张文章表.包含字段`id-摘要-内容`.而系统中的展示形式是刷新出一个列表,列表中仅包含标题和摘要,当用户点击某篇文章进入详情时才需要正文内容.此时,如果数据量大,将内容这个很大且不经常使用的列放在一起会拖慢原表的查询速度.我们可以将上面的表分为两张.`id-摘要`,`id-内容`.当用户点击详情,那主键再来取一次内容即可.而增加的存储量只是很小的主键字段.代价很小.

当然,分表其实和业务的关联度很高,在分表之前一定要做好调研以及 benchmark.不要按照自己的猜想盲目操作.

7. 什么是存储过程？有哪些优缺点？

存储过程是一些预编译的 SQL 语句。1、更加直白的理解：存储过程可以说是一个记录集，它是由一些 T-SQL 语句组成的代码块，这些 T-SQL 语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。2、存储过程是一个预编译的代码块，执行效率比较高,一个存储过程替代大量 T_SQL 语句 ，可以降低网络通信量，提高通信速率,可以一定程度上确保数据安全

但是,在互联网项目中,其实是不太推荐存储过程的,比较出名的就是阿里的《Java 开发手册》中禁止使用存储过程,我个人的理解是,在互联网项目中,迭代太快,项目的生命周期也比较短,人员流动相比于传统的项目也更加频繁,在这样的情况下,存储过程的管理确实是没有那么方便,同时,复用性也没有写在服务层那么好.

8. 说一说三个范式

第一范式: 每个列都不可以再拆分. 第二范式: 非主键列完全依赖于主键,而不能是依赖于主键的一部分. 第三范式: 非主键列只依赖于主键,不依赖于其他非主键.

在设计数据库结构的时候,要尽量遵守三范式,如果不遵守,必须有足够的理由.比如性能. 事实上我们经常会为了性能而妥协数据库的设计.

9. MyBatis 中的#

乱入了一个奇怪的问题.....我只是想单独记录一下这个问题,因为出现频率太高了.

‘#’ 会将传入的内容当做字符串,而有什么区别?* * 乱入了一个奇怪的问题..... 我只是想单独记录一下这个问题,因为出现频率太高了.#会将传入的内容当做字符串,而会直接将传入值拼接在 sql 语句中.

所以#可以在一定程度上预防 sql 注入攻击.