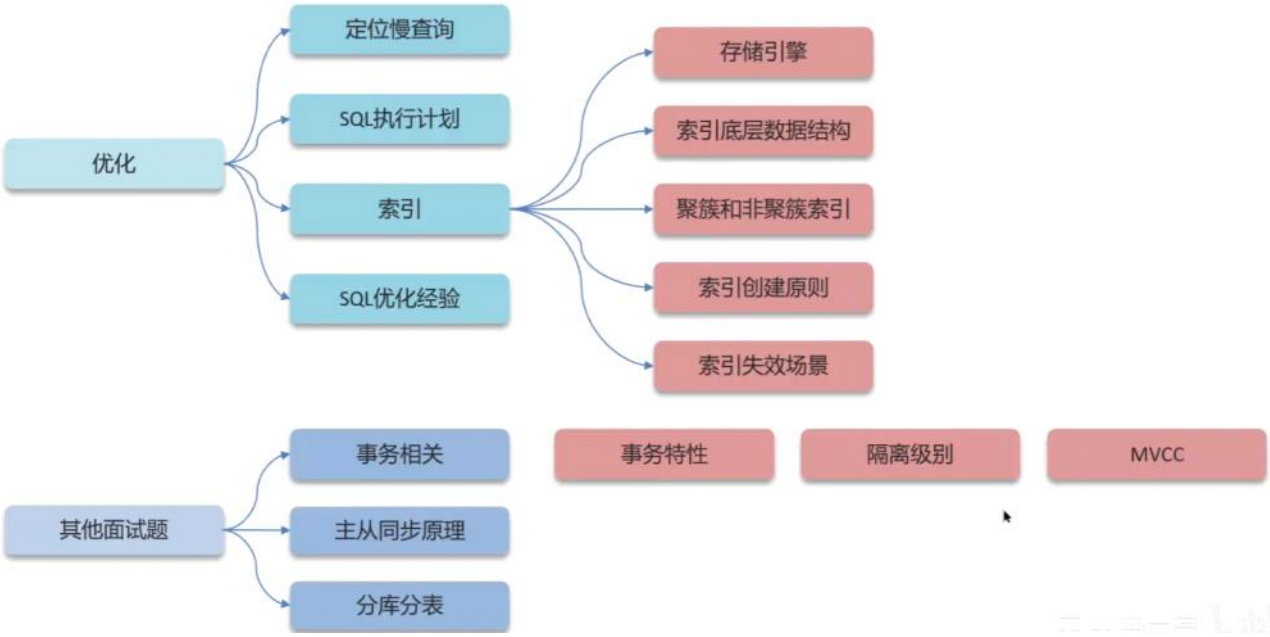
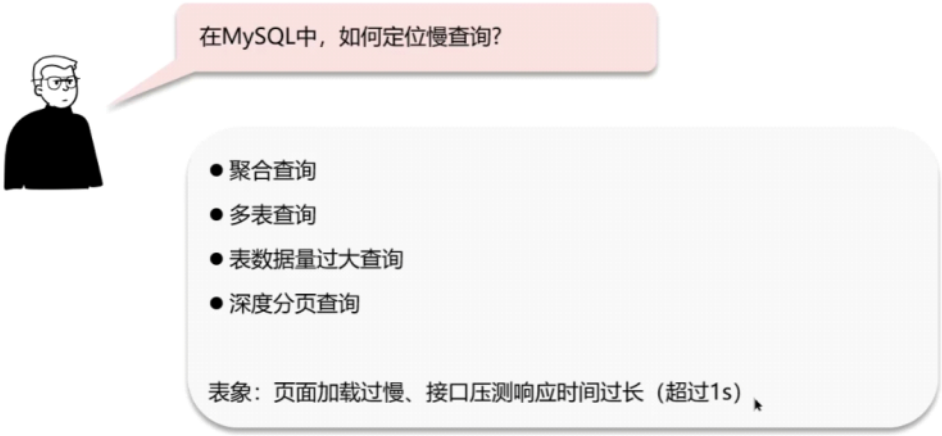
优化、B+树、索引

2024年3月9日 19:23

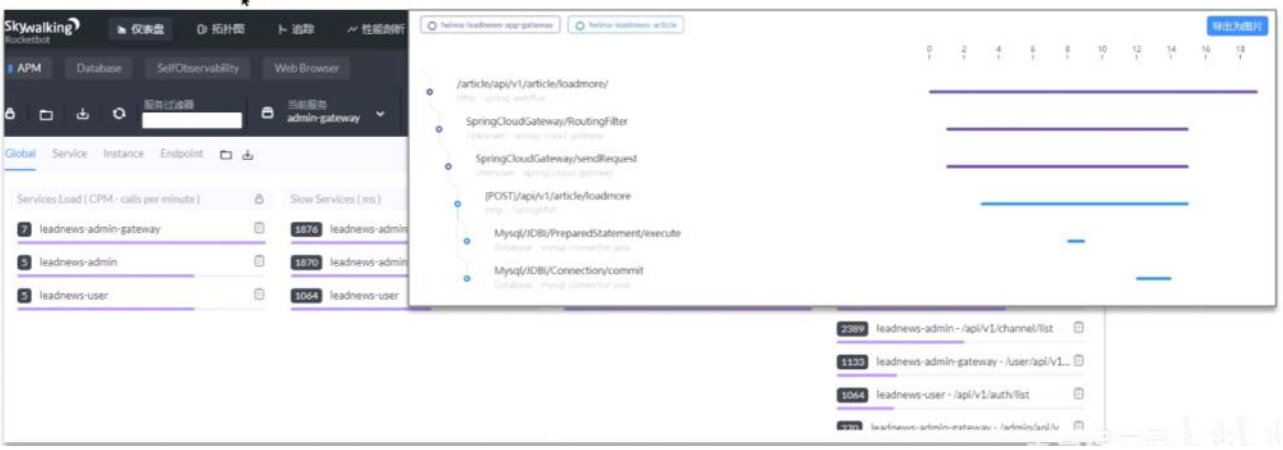




如何定位慢查询？

方案一：开源工具

- 调试工具：Arthas
- 运维工具：Prometheus 、Skywalking



如何定位慢查询？

方案二：MySQL自带慢日志

慢查询日志记录了所有执行时间超过指定参数（long_query_time，单位：秒，默认10秒）的所有SQL语句的日志

如果要开启慢查询日志，需要在MySQL的配置文件（/etc/my.cnf）中配置如下信息：

```
# 开启MySQL慢日志查询开关
slow_query_log=1
# 设置慢日志的时间为2秒，SQL语句执行时间超过2秒，就会视为慢查询，记录慢查询日志
long_query_time=2
```

配置完毕之后，通过以下指令重新启动MySQL服务器进行测试，查看慢日志文件中记录的信息/var/lib/mysql/localhost-slow.log。

```
# Time: 2023-03-15T15:21:55.178101Z
# User@Host: root[root] @ localhost [::1]  Id:     3
# Query_time: 45.472697  Lock_time: 0.003903 Rows_sent: 10000000  Rows_examined: 10000000
use db01;
SET timestamp=1678893715;
select * from tb_sku;
```

如何定位慢查询?

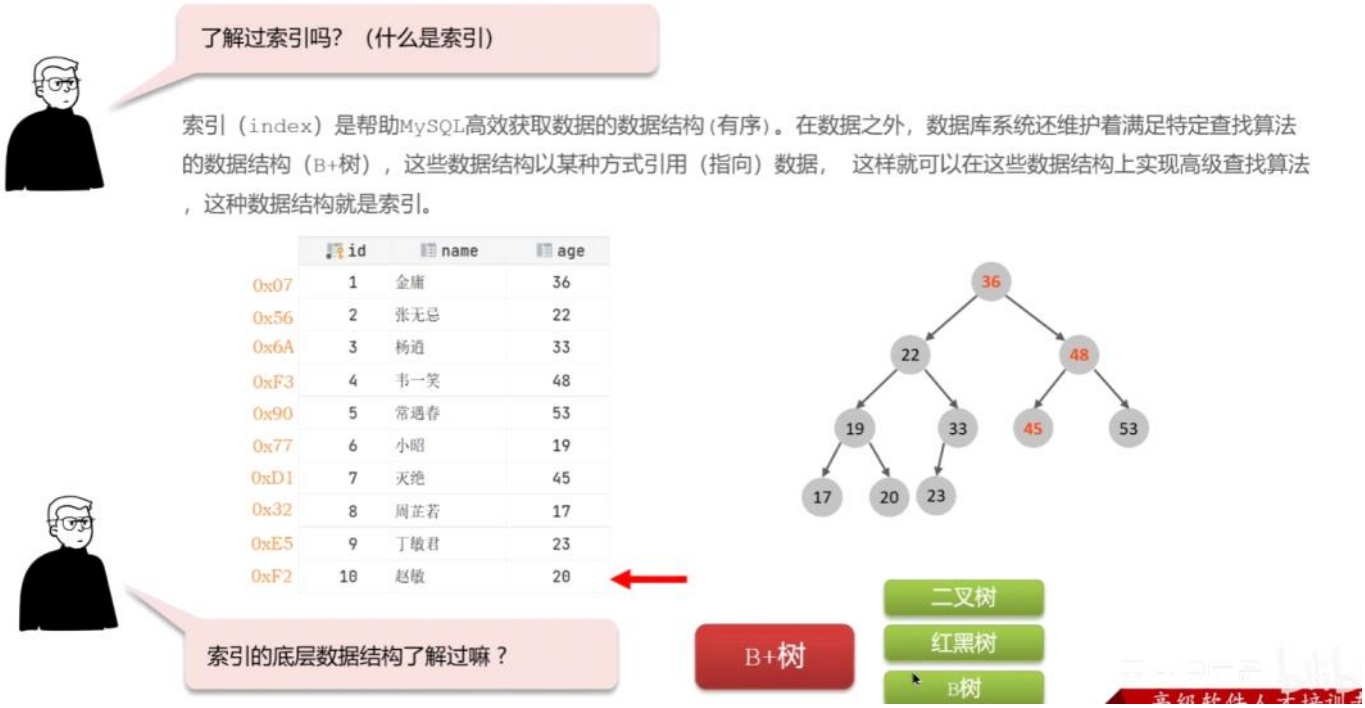
1. 介绍一下当时产生问题的场景（我们当时的一个接口测试的时候非常的慢，压测的结果大概5秒钟）
2. 我们系统中当时采用了运维工具（Skywalking），可以监测出哪个接口，最终因为是sql的问题
3. 在mysql中开启了慢日志查询，我们设置的值就是2秒，一旦sql执行超过2秒就会记录到日志中（调试阶段）

面试官：MySQL中，如何定位慢查询?

候选人：

嗯~，我们当时做压测的时候有的接口非常的慢，接口的响应时间超过了2秒以上，因为我们当时的系统部署了运维的监控系统Skywalking，在展示的报表中可以看到是哪一个接口比较慢，并且可以分析这个接口哪部分比较慢，这里可以看到SQL的具体的执行时间，所以可以定位是哪个sql出了问题

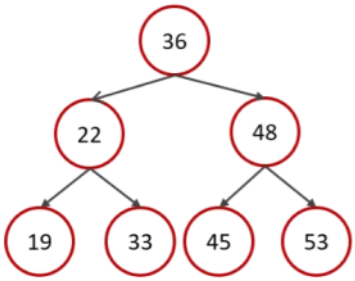
如果，项目中没有这种运维的监控系统，其实在MySQL中也提供了慢日志查询的功能，可以在MySQL的系统配置文件中开启这个慢日志的功能，并且也可以设置SQL执行超过多少时间来记录到一个日志文件中，我记得上一个项目配置的是2秒，只要SQL执行的时间超过了2秒就会记录到日志文件中，我们就可以在日志文件找到执行比较慢的SQL了。



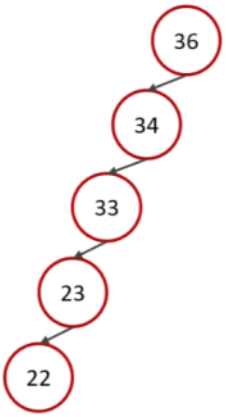
数据结构对比

MySQL默认使用的索引底层数据结构是B+树。再聊B+树之前，我们先聊聊二叉树和B树

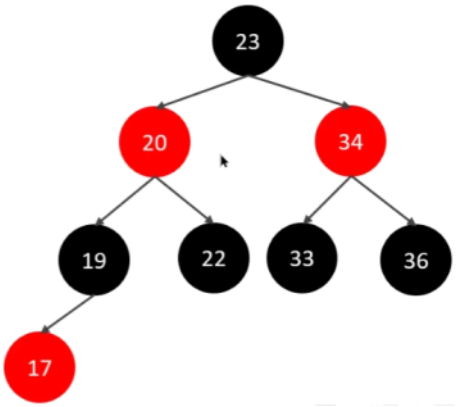
二叉搜索树



最坏的二叉树



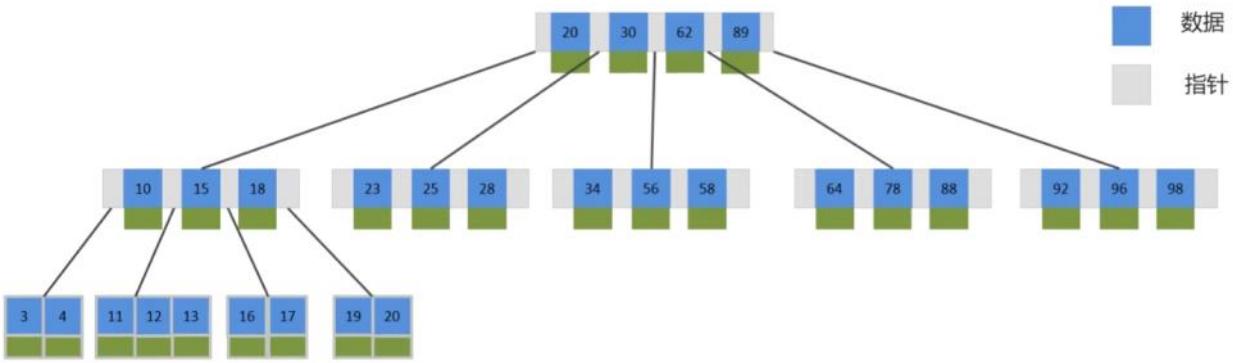
红黑树



数据结构对比

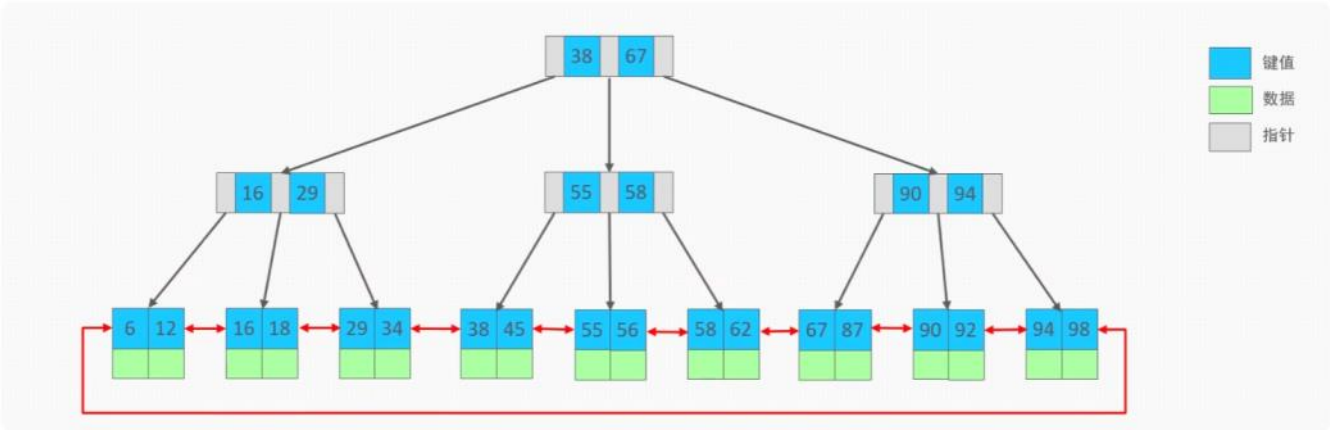
B-Tree，B树是一种多叉路衡查找树，相对于二叉树，B树每个节点可以有多个分支，即多叉。

以一颗最大度数（max-degree）为5(5阶)的b-tree为例，那这个B树每个节点最多存储4个key



数据结构对比

B+Tree是在BTree基础上的一种优化，使其更适合实现外存储索引结构，InnoDB存储引擎就是用B+Tree实现其索引结构



B树与B+树对比:

①：磁盘读写代价B+树更低；②：查询效率B+树更加稳定；③：B+树便于扫库和区间查询

了解过索引吗？（什么是索引）

- 索引（index）是帮助MySQL高效获取数据的数据结构(有序)
- 提高数据检索的效率，降低数据库的IO成本（不需要全表扫描）
- 通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

索引的底层数据结构了解过嘛？

MySQL的InnoDB引擎采用的B+树的数据结构来存储索引

- 阶数更多，路径更短
- 磁盘读写代价B+树更低，非叶子节点只存储指针，叶子阶段存储数据
- B+树便于扫库和区间查询，叶子节点是一个双向链表

面试官： 了解过索引吗？（什么是索引）

候选人： 嗯，索引在项目中还是比较常见的，它是帮助MySQL高效获取数据的数据结构，主要是用来提高数据检索的效率，降低数据库的IO成本，同时通过索引列对数据进行排序，降低数据排序的成本，也能降低了CPU的消耗

面试官： 索引的底层数据结构了解过嘛？

候选人： MySQL的默认的存储引擎InnoDB采用的B+树的数据结构来存储索引，选择B+树的主要的原因是：第一阶数更多，路径更短，第二个磁盘读写代价B+树更低，非叶子节点只存储指针，叶子阶段存储数据，第三是B+树便于扫库和区间查询，叶子节点是一个双向链表

面试官： B树和B+树的区别是什么呢？

候选人： 第一：在B树中，非叶子节点和叶子节点都会存放数据，而B+树的所有的数据都会出现在叶子节点，在查询的时候，B+树查找效率更加稳定

第二：在进行范围查询的时候，B+树效率更高，因为B+树都在叶子节点存储，并且叶子节点是一个双向链表

什么是聚簇索引什么是非聚簇索引？

什么是聚集索引，什么是二级索引（非聚集索引）

什么是回表?

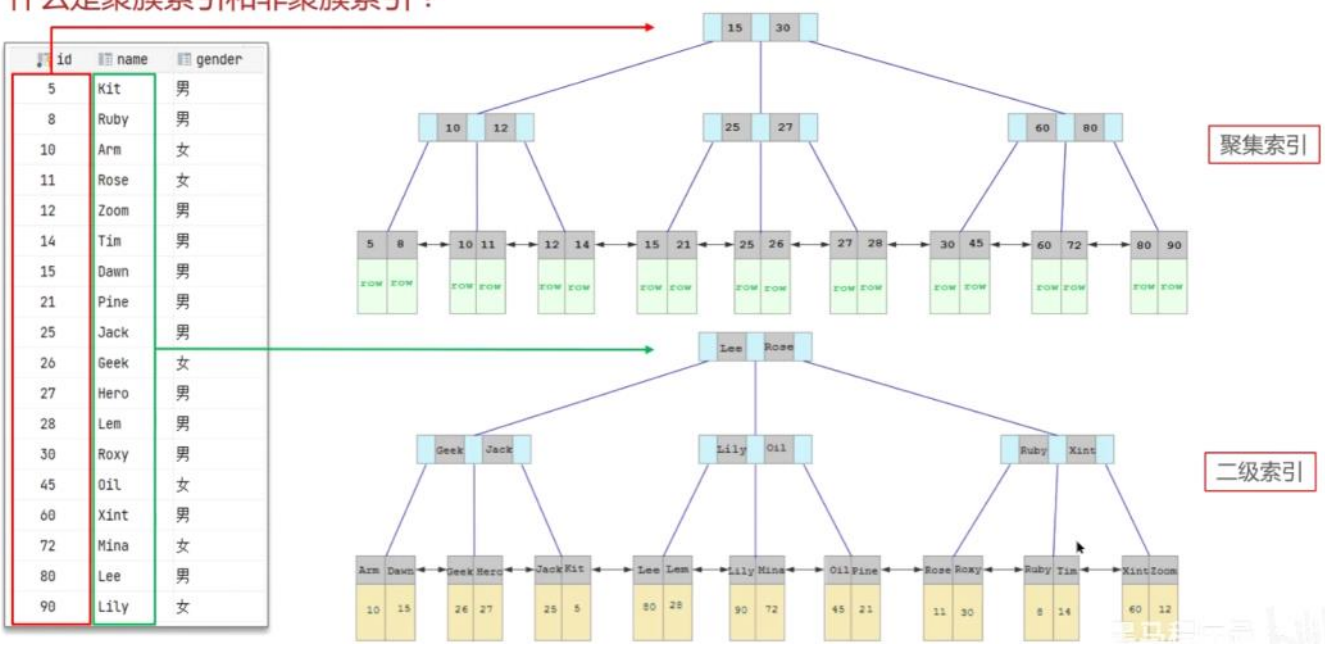
id	name	gender
5	Kit	男
8	Ruby	男
10	Arm	女
11	Rose	女
12	Zoom	男
14	Tim	男
15	Dawn	男
21	Pine	男
25	Jack	男
26	Geek	女
27	Hero	男
28	Lem	男
30	Roxy	男
45	Oil	女
60	Xint	男
72	Mina	女
80	Lee	男
90	Lily	女

分类	含义	特点
聚集索引(Clustered Index)	将数据存储与索引放到了一块，索引结构的叶子节点保存了行数据	必须有,而且只有一个
二级索引(Secondary Index)	将数据与索引分开存储，索引结构的叶子节点关联的是对应的主键	可以存在多个

聚集索引选取规则:

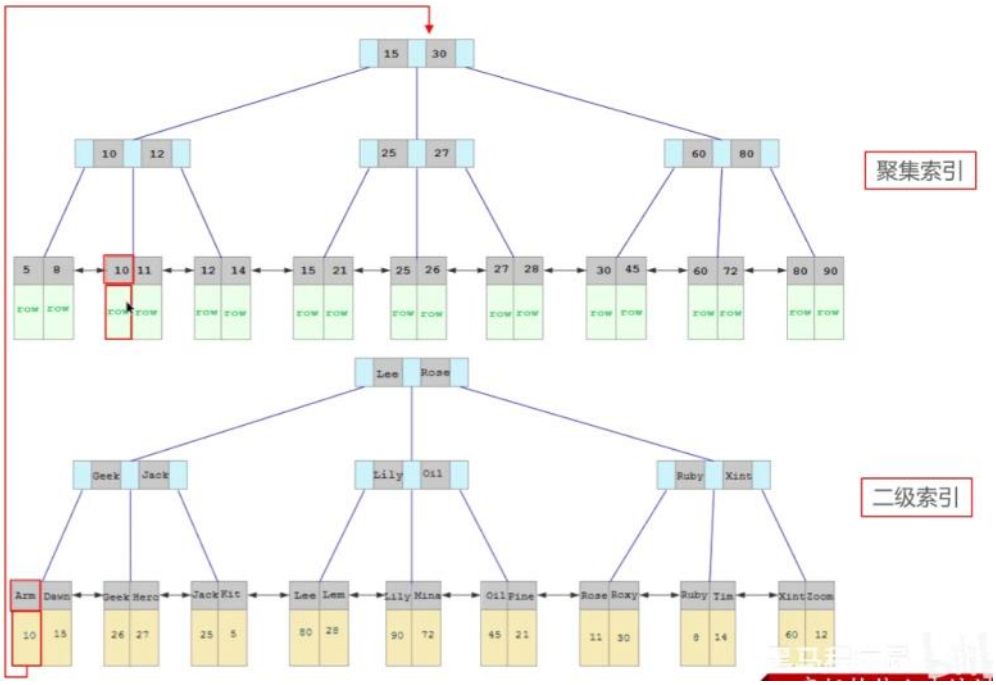
- 如果存在主键，主键索引就是聚集索引。
- 如果不存在主键，将使用第一个唯一（UNIQUE）索引作为聚集索引。
- 如果表没有主键，或没有合适的唯一索引，则InnoDB会自动生成一个rowid作为隐藏的聚集索引。

什么是聚簇索引和非聚簇索引？



回表查询

```
select * from user where name = 'Arm';
```



什么是聚簇索引什么是非聚簇索引？

- 聚簇索引（聚集索引）：数据与索引放到一块，B+树的叶子节点保存了整行数据，有且只有一个
- 非聚簇索引（二级索引）：数据与索引分开存储，B+树的叶子节点保存对应的主键，可以有多个

知道什么是回表查询嘛？

通过二级索引找到对应的主键值，到聚集索引中查找整行数据，这个过程就是回表

面试官：什么是聚簇索引什么是非聚簇索引？

候选人：

好的~，聚簇索引主要是指数据与索引放到一块，B+树的叶子节点保存了整行数据，有且只有一个，一般情况下主键在作为聚簇索引的

非聚簇索引值的是数据与索引分开存储，B+树的叶子节点保存对应的主键，可以有多个，一般我们自己定义的索引都是非聚簇索引

面试官：知道什么是回表查询嘛？

候选人：嗯，其实跟刚才介绍的聚簇索引和非聚簇索引是有关系的，回表的意思就是通过二级索引找到对应的主键值，然后再通过主键值找到聚集索引中所对应的整行数据，这个过程就是回表

【备注：如果面试官直接问回表，则需要先介绍聚簇索引和非聚簇索引】

知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到 。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

- id为主键，默认是主键索引
- name字段为普通索引

```
select * from tb_user where id = 1                          覆盖索引
select id，name from tb_user where name = 'Arm'             覆盖索引
select id，name，gender from tb_user where name = 'Arm'     非覆盖索引(需要回表查询)
```

覆盖索引

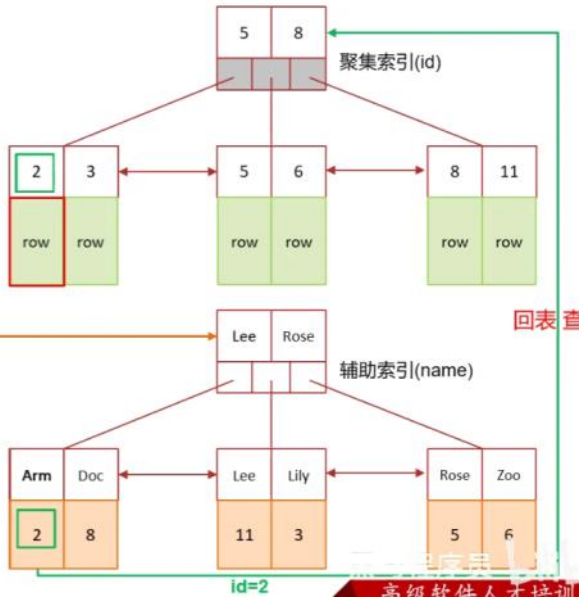
覆盖索引是指 查询使用了索引，并且需要返回的列，在该索引中已经全部能够找到 。

id	name	gender	createdate
2	Arm	1	2021-01-01
3	Lily	0	2021-05-01
5	Rose	0	2021-02-14
6	Zoo	1	2021-06-01
8	Doc	1	2021-03-08
11	Lee	1	2020-12-03

```
select * from tb_user where id = 2;

select id, name from tb_user where name = 'Arm';

select id,name,gender from tb_user where name = 'Arm';
```



知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，返回的列，必须在索引中全部能够找到

- 使用id查询，直接走聚集索引查询，一次索引扫描，直接返回数据，性能高。
- 如果返回的列中没有创建索引，有可能会触发回表查询，尽量避免使用select *

MYSQL超大分页怎么处理？

可以使用覆盖索引解决

MYSQL超大分页处理

在数据量比较大时，如果进行limit分页查询，在查询时，越往后，分页查询效率越低。

我们一起来看看执行limit分页查询耗时对比：

```
mysql> select * from tb_sku limit 0,10;
10 rows in set (0.00 sec)

mysql> select * from tb_sku limit 9000000,10;
10 rows in set (11.05 sec)
```

因为，当在进行分页查询时，如果执行 limit 9000000,10 ，此时需要MySQL排序前9000010 记录，仅仅返回9000000 - 9000010 的记录，其他记录丢弃，查询排序的代价非常大 。

MYSQL超大分页处理

优化思路: 一般分页查询时，通过创建 覆盖索引 能够比较好地提高性能，可以通过覆盖索引加子查询形式进行优化

```
select *
from tb_sku t,
    (select id from tb_sku order by id limit 9000000,10) a
where t.id = a.id;
```

```
10 rows in set (7.13 sec)
```

```
10 rows in set (7.37 sec)
```

```
10 rows in set (7.19 sec)
```

知道什么叫覆盖索引嘛？

覆盖索引是指查询使用了索引，返回的列，必须在索引中全部能够找到

- 使用id查询，直接走聚集索引查询，一次索引扫描，直接返回数据，性能高。
- 如果返回的列中没有创建索引，有可能会触发回表查询，尽量避免使用select *

MYSQL超大分页怎么处理？

问题：在数据量比较大时，limit分页查询，需要对数据进行排序，效率低

解决方案：覆盖索引+子查询

索引创建原则有哪些?

- 先陈述自己在实际的工作中是怎么用的
- 主键索引
- 唯一索引
- 根据业务创建的索引(复合索引)

索引创建原则有哪些?

1). 针对于数据量较大，且查询比较频繁的表建立索引。 单表超过**10万**数据（增加用户体验）

2). 针对于常作为查询条件（where）、排序（order by）、分组（group by）操作的字段建立索引。

3). 尽量选择区分度高的列作为索引，尽量建立唯一索引，区分度越高，使用索引的效率越高。

4). 如果是字符串类型的字段，字段的长度较长，可以针对于字段的特点，建立前缀索引。

5). 尽量使用联合索引，减少单列索引，查询时，联合索引很多时候可以覆盖索引，节省存储空间，避免回表，提高查询效率。

6). 要控制索引的数量，索引并不是多多益善，索引越多，维护索引结构的代价也就越大，会影响增删改的效率。

7). 如果索引列不能存储NULL值，请在创建表时使用NOT NULL约束它。当优化器知道每列是否包含NULL值时，它可以更好地确定哪个索引最有效地用于查询。

索引创建原则有哪些?

1). 数据量较大，且查询比较频繁的表 重要

2). 常作为查询条件、排序、分组的字段 重要

3). 字段内容区分度高

4). 内容较长，使用前缀索引

5). 尽量联合索引 重要

6). 要控制索引的数量 重要

7). 如果索引列不能存储NULL值，请在创建表时使用NOT NULL约束它

面试官：索引创建原则有哪些?

候选人：嗯，这个情况有很多，不过都有一个大前提，就是表中的数据要超过10万以上，我们才会创建索引，并且添加索引的字段是查询比较频繁的字段，一般也是像作为查询条件，排序字段或分组的字段这些。

还有就是，我们通常创建索引的时候都是使用复合索引来创建，一条sql的返回值，尽量使用覆盖索引，如果字段的区分度不高的话，我们也会把它放在组合索引后面的字段。

如果某一个字段的内容较长，我们会考虑使用前缀索引来使用，当然并不是所有的字段都要添加索引，这个索引的数量也要控制，因为添加索引也会导致新增改的速度变慢。

什么情况下索引会失效？

索引失效的情况有很多，可以说一些自己遇到过的，不要张口就得得得说一堆背诵好的面试题

（适当的思考一下，回想一下，更真实）

给tb_seller创建联合索引，字段顺序：name，status，address

```
mysql> show index from tb_seller;
+-----------+------------+-----------------+--------------+-------------+
| Table     | Non_unique | Key_name        | Seq_in_index | Column_name |
+-----------+------------+-----------------+--------------+-------------+
| tb_seller |          1 | tb_seller_index |            1 | name        |
| tb_seller |          1 | tb_seller_index |            2 | status      |
| tb_seller |          1 | tb_seller_index |            3 | address     |
+-----------+------------+-----------------+--------------+-------------+
```

那快读判断索引是否失效了呢？ 执行计划explain

什么情况下索引会失效？

```
mysql> show index from tb_seller;
+-----------+------------+-----------------+--------------+-------------+
| Table     | Non_unique | Key_name        | Seq_in_index | Column_name |
+-----------+------------+-----------------+--------------+-------------+
| tb_seller |          1 | tb_seller_index |            1 | name        |
| tb_seller |          1 | tb_seller_index |            2 | status      |
| tb_seller |          1 | tb_seller_index |            3 | address     |
+-----------+------------+-----------------+--------------+-------------+
```

1). 违反最左前缀法则

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始，并且不跳过索引中的列。匹配最左前缀法则，走索引：

```
mysql> explain select * from tb_seller where name = '小米科技';
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys   | key             | key_len | ref   | rows | filtered | Extra |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 303     | const |    1 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from tb_seller where name = '小米科技' and status = '1';
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys   | key             | key_len | ref         | rows | filtered | Extra |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 309     | const,const |    1 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from tb_seller where name = '小米科技' and status = '1' and address = '北京市';
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys   | key             | key_len | ref               | rows | filtered | Extra |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------------+------+----------+-------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 612     | const,const,const |    1 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+-----------------+-----------------+---------+-------------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

什么情况下索引会失效？

```
mysql> show index from tb_seller;
+-----------+------------+----------------+--------------+-------------+
| Table     | Non_unique | Key_name       | Seq_in_index | Column_name |
+-----------+------------+----------------+--------------+-------------+
| tb_seller |          1 | tb_seller_index|            1 | name        |
| tb_seller |          1 | tb_seller_index|            2 | status      |
| tb_seller |          1 | tb_seller_index|            3 | address     |
+-----------+------------+----------------+--------------+-------------+
```

违法最左前缀法则， 索引失效:

```
mysql> explain select * from tb_seller where status = '1' and address = '北京市';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | tb_seller | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   12 |     8.33 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from tb_seller where status = '1';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | tb_seller | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   12 |    10.00 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

如果符合最左法则，但是出现跳跃某一列，只有最左列索引生效:

```
mysql> explain select * from tb_seller where name = '小米科技' and address = '北京市';
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------+------+----------+-----------------------+
| id | select_type | table     | partitions | type | possible_keys | key           | key_len | ref   | rows | filtered | Extra                 |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------+------+----------+-----------------------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 303     | const |    1 |    10.00 | Using index condition |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------+------+----------+-----------------------+
```

什么情况下索引会失效？

2). 范围查询右边的列，不能使用索引 。

```
mysql> explain select * from tb_seller where name = '小米科技' and status = '1' and address = '北京市';
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key           | key_len | ref               | rows | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------------+------+----------+-------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 612     | const,const,const |    1 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+---------------+---------------+---------+-------------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from tb_seller where name = '小米科技' and status > '1' and address = '北京市';
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys | key           | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | tb_seller | NULL       | range | tb_seller_index | tb_seller_index | 309     | NULL |    1 |    10.00 | Using index condition |
+----+-------------+-----------+------------+-------+---------------+---------------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

根据前面的两个字段 name， status 查询是走索引的， 但是最后一个条件address 没有用到索引。

什么情况下索引会失效？

3). 不要在索引列上进行运算操作， 索引将失效。

```
mysql> select * from tb_seller where substring(name,3,2) = '科技';
+----------+--------------------+--------------+----------------------------------+--------+---------+---------------------+
| sellerid | name               | nickname     | password                         | status | address | createtime          |
+----------+--------------------+--------------+----------------------------------+--------+---------+---------------------+
| baidu    | 百度科技有限公司   | 百度小店     | e10adc3949ba59abbe56e057f20f883e | 1      | 北京市  | 2088-01-01 12:00:00 |
| huawei   | 华为科技有限公司   | 华为小店     | e10adc3949ba59abbe56e057f20f883e | 0      | 北京市  | 2088-01-01 12:00:00 |
| luoji    | 罗技科技有限公司   | 罗技小店     | e10adc3949ba59abbe56e057f20f883e | 1      | 北京市  | 2088-01-01 12:00:00 |
| ourpalm  | 掌趣科技股份有限公司 | 掌趣小店   | e10adc3949ba59abbe56e057f20f883e | 1      | 北京市  | 2088-01-01 12:00:00 |
| qiandu   | 千度科技           | 千度小店     | e10adc3949ba59abbe56e057f20f883e | 2      | 北京市  | 2088-01-01 12:00:00 |
| sina     | 新浪科技有限公司   | 新浪官方旗舰店 | e10adc3949ba59abbe56e057f20f883e | 1      | 北京市  | 2088-01-01 12:00:00 |
| xiaomi   | 小米科技           | 小米官方旗舰店 | e10adc3949ba59abbe56e057f20f883e | 1      | 西安市  | 2088-01-01 12:00:00 |
+----------+--------------------+--------------+----------------------------------+--------+---------+---------------------+
7 rows in set (0.00 sec)

mysql> explain select * from tb_seller where substring(name,3,2) = '科技';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | tb_seller | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   12 |   100.00 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

什么情况下索引会失效？

4). 字符串不加单引号，造成索引失效。

```
mysql> explain select * from tb_seller where name = '科技' and status = '0';
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------------+------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys  | key            | key_len | ref         | rows | filtered | Extra |
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------------+------+----------+-------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 309     | const,const |    1 |   100.00 | NULL  |
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from tb_seller where name = '科技' and status = 0;
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------+------+----------+-----------------------+
| id | select_type | table     | partitions | type | possible_keys  | key            | key_len | ref   | rows | filtered | Extra                 |
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------+------+----------+-----------------------+
|  1 | SIMPLE      | tb_seller | NULL       | ref  | tb_seller_index | tb_seller_index | 303     | const |    1 |    10.00 | Using index condition |
+----+-------------+-----------+------------+------+----------------+----------------+---------+-------+------+----------+-----------------------+
1 row in set, 2 warnings (0.00 sec)
```

由于，在查询是，没有对字符串加单引号， MySQL的查询优化器，会自动的进行类型转换，造成索引失效。

什么情况下索引会失效？

5).以%开头的Like模糊查询，索引失效。如果仅仅是尾部模糊匹配，索引不会失效。如果是头部模糊匹配，索引失效。

```
mysql> explain select sellerid,name from tb_seller where name like '%黑马程序员%';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | tb_seller | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   12 |    11.11 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select sellerid,name from tb_seller where name like '%黑马程序员';
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | tb_seller | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   12 |    11.11 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select sellerid,name from tb_seller where name like '黑马程序员%';
+----+-------------+-----------+------------+-------+----------------+----------------+---------+------+------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys  | key            | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-----------+------------+-------+----------------+----------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | tb_seller | NULL       | range | tb_seller_index | tb_seller_index | 303     | NULL |    1 |   100.00 | Using index condition |
+----+-------------+-----------+------------+-------+----------------+----------------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

什么情况下索引会失效？

① 违反最左前缀法则

② 范围查询右边的列，不能使用索引

③ 不要在索引列上进行运算操作， 索引将失效

④ 字符串不加单引号，造成索引失效。(类型转换)

⑤ 以%开头的Like模糊查询，索引失效

谈一谈你对sql的优化的经验

- 表的设计优化
- 索引优化　参考优化创建原则和索引失效
- SQL语句优化
- 主从复制、读写分离
- 分库分表　后面有专门章节介绍

谈谈你对sql的优化的经验

- **表的设计优化（参考阿里开发手册《嵩山版》）**

① 比如设置合适的数值（tinyint　int　bigint），要根据实际情况选择

② 比如设置合适的字符串类型（char和varchar）char定长效率高，varchar可变长度，效率稍低

- **SQL语句优化**

① SELECT语句务必指明字段名称（避免直接使用select * ）

② SQL语句要避免造成索引失效的写法

③ 尽量用union all代替union　union会多一次过滤，效率低

④ 避免在where子句中对字段进行表达式操作

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 1000; j++) {

    }
}
```

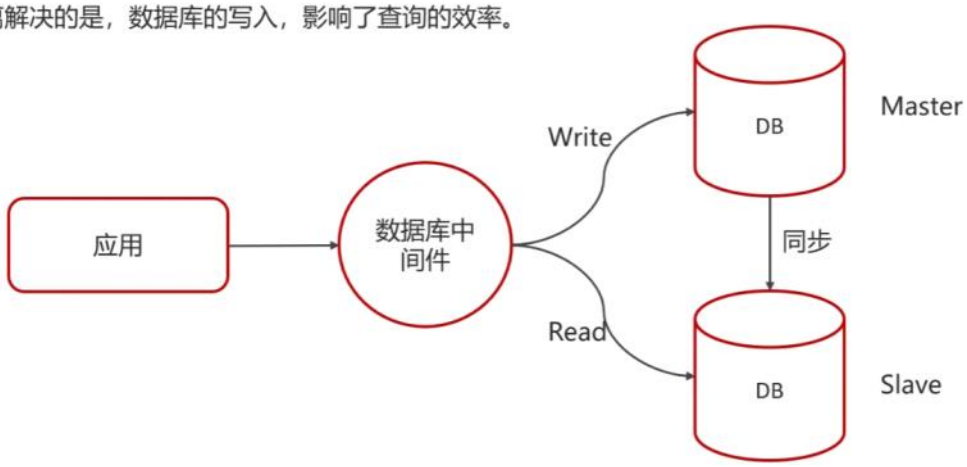
⑤ Join优化 能用innerjoin 就不用left join right join，如必须使用 一定要以小表为驱动，
内连接会对两个表进行优化，优先把小表放到外边，把大表放到里边。left join 或 right join，不会重新调整顺序

谈谈你对sql的优化的经验

- **主从复制、读写分离**

如果数据库的使用场景读的操作比较多的时候，为了避免写的操作所造成的性能影响 可以采用读写分离的架构。

读写分离解决的是，数据库的写入，影响了查询的效率。



面试官：sql的优化的经验

候选人：嗯，这个在项目还是挺常见的，当然如果直说sql优化的话，我们会从这几方面考虑，比如

建表的时候、使用索引、sql语句的编写、主从复制，读写分离，还有一个是如果量比较大的话，可以考虑分库分表

面试官：创建表的时候，你们是如何优化的呢？

候选人：这个我们主要参考的阿里出的那个开发手册《嵩山版》，就比如，在定义字段的时候需要结合字段的内容来选择合适的类型，如果是数值的话，像tinyint、int 、bigint这些类型，要根据实际情况选择。如果是字符串类型，也是结合存储的内容来选择char和varchar或者text类型

面试官：那在使用索引的时候，是如何优化呢？

候选人：【参考索引创建原则 进行描述】

面试官：你平时对sql语句做了哪些优化呢？

候选人：嗯，这个也有很多，比如SELECT语句务必指明字段名称，不要直接使用select *，还有就是要注意SQL语句避免造成索引失效的写法；如果是聚合查询，尽量用union all代替union ，union会多一次过滤，效率比较低；如果是表关联的话，尽量使用innerjoin ，不要使用用left join right join，如必须使用 一定要以小表为驱动

那这个SQL语句执行很慢，如何分析呢？

- 聚合查询
- 多表查询
- 表数据量过大查询
- 深度分页查询

SQL执行计划（找到慢的原因）

SELECT COUNT(*) FROM students

Select * from students,teachers

一个SQL语句执行很慢，如何分析

可以采用EXPLAIN 或者 DESC命令获取 MySQL 如何执行 SELECT 语句的信息

语法：

- 直接在select语句之前加上关键字 explain / desc

```
EXPLAIN  SELECT  字段列表   FROM  表名   WHERE  条件 ;
```

```
mysql> explain select * from t_user where id = '1';
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t_user | NULL       | const | PRIMARY       | PRIMARY | 98      | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

一个SQL语句执行很慢，如何分析

```
mysql> explain select * from t_user where id = '1';
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t_user | NULL       | const | PRIMARY       | PRIMARY | 98      | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

- possible_key 当前sql可能会使用到的索引
- key 当前sql实际命中的索引
- key_len 索引占用的大小

通过它们两个查看是否可能会命中索引

- Extra 额外的优化建议

Extra	含义
Using where; Using Index	查找使用了索引，需要的数据都在索引列中能找到，不需要回表查询数据
Using index condition	查找使用了索引，但是需要回表查询数据

一个SQL语句执行很慢，如何分析

```
mysql> explain select * from t_user where id = '1';
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t_user | NULL       | const | PRIMARY       | PRIMARY | 98      | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

- possible_key 当前sql可能会使用到的索引
- key 当前sql实际命中的索引
- key_len 索引占用的大小

通过它们两个查看是否可能会命中索引

- Extra 额外的优化建议

Extra	含义
Using where; Using Index	查找使用了索引，需要的数据都在索引列中能找到，不需要回表查询数据
Using index condition	查找使用了索引，但是需要回表查询数据

一个SQL语句执行很慢，如何分析

```
mysql> explain select * from t_user where id = '1';
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t_user | NULL       | const | PRIMARY       | PRIMARY | 98      | const |    1 |   100.00 | NULL  |
+----+-------------+--------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

- type 这条sql的连接的类型，性能由好到差为NULL、system、const、eq_ref、ref、range、 index、all
  - system：查询系统中的表
  - const：根据主键查询
  - eq_ref：主键索引查询或唯一索引查询
  - ref：索引查询
  - range：范围查询
  - index：索引树扫描
  - all：全盘扫描

那这个SQL语句执行很慢，如何分析呢?

可以采用MySQL自带的分析工具 EXPLAIN

- 通过key和key_len检查是否命中了索引（索引本身存在是否有失效的情况）
- 通过type字段查看sql是否有进一步的优化空间，是否存在全索引扫描或全盘扫描
- 通过extra建议判断，是否出现了回表的情况，如果出现了，可以尝试添加索引或修改返回字段来修复

面试官：那这个SQL语句执行很慢，如何分析呢？

候选人：如果一条sql执行很慢的话，我们通常会使用mysql自动的执行计划explain来去查看这条sql的执行情况，比如在这里面可以通过key和key_len检查是否命中了索引，如果本身已经添加了索引，也可以判断索引是否有失效的情况，第二个，可以通过type字段查看sql是否有进一步的优化空间，是否存在全索引扫描或全盘扫描，第三个可以通过extra建议来判断，是否出现了回表的情况，如果出现了，可以尝试添加索引或修改返回字段来修复