全网最全数据分析师面试干货-sql篇

1高频考点

1）sql开窗函数/窗口函数、row_number、rank、dense_rank三个函数排序的区别

（详细内容可以关注@赵小洛洛洛公众号）

窗口函数的主要作用是对数据进行分组排序、求和、求平均值、计数等。在实际工作中被广泛应用，可以大大提高数据查询效率。

基本语法：

```
<窗口函数>OVER ([partition by <列名>] order by <列名>)
```

其中，partition by划分的范围被称为窗口。而order by决定着窗口范围内的数据以什么样的方式排序。

OVER子句用于为行为定义一个窗口（windows），以便进行特定的运算。可以把行的窗口简单地认为是运算将要操作的一个行的集合。由于OVER子句为这些函数提供了一个行的窗口，所以这些函数也称之为开窗函数。

窗口函数与group by的区别：

- **两个order by的区别**，第一个窗口函数中的order by只是决定着窗口里的数据的排序方式，第二个普通的order by决定查询出的数据以什么样的方式整体排序；
- 窗口函数可以在保留原表中的全部数据之后，可以对某些字段做分组排序或者计算，而group by只能保留与分组字段聚合的结果；
- 在加入窗口函数的基础上SQL的执行顺序也会发生变化，具体的执行顺序如下（window就是窗口函数）；

2）表关联：left join 、right join、join

sql中的表连接/关联查询有inner join(内连接）、left join(左连接)、right join（右连接）、full join（全连接）四种方式。

其中，left /right join 是left /rightouter join简写，left /right join默认都是outer属性的 。
inner join（内连接）：在两张表进行连接查询时，只保留两张表中完全匹配的结果集。

left join(左连接)：在两张表进行连接查询时，会返回左表所有的行，而右表中没有匹配的记录则会表示为null。

right join（右连接）：在两张表进行连接查询时，会返回由表所有的行，而左表中没有匹配的记录则会表示为null。

full join（全连接）：在两张表进行连接查询时，返回左表和右表中所有没有匹配的行。

3）having、where

where和having的区别：

where是一个约束声明，使用where来约束来自数据库的数据；where是在结果返回之前起作用的；where中不能使用聚合函数。

having是一个过滤声明；在查询返回结果集以后，对查询结果进行的过滤操作；在having中可以使用聚合函数。

对一组(多条)数据操作的函数，需要配合group by 来使用。

where和having的执行顺序：

2 条件子句（where）

2.1 比较运算符（适用于区间）

比较运算符包括=（等于），>=（大于等于），<=（小于等于），!=（不等于），>（大于），<（小于）。

例如：查询年龄sage小于30的学生

```
where sage < 30
```

2.2 确定范围（适用于连续范围）

between ... and ...为取值限定了一个范围。

例如：查询年龄大于等于10小于等于20的学生

```
where sage between 10 and 20
```

2.3 确定集合（适用于离散的少数值）

例如：插入年龄为10,20,30的学生

```
where sage in (10,20,30)
```

**in可以和not一起使用，表示不在这个区间的值

```
**where sage not in (10,20,30)
```

2.4 字符匹配（模糊查询）

通过like关键字和正则表达式匹配，常用的通配符有%（任意个字符）和_（一个字符）。

例如：查询名字sname带“王”的学生

```
where sname like “%王%”
```

2.5 判断是否为空值

通过is null关键字判断值是否为空。

例如：查询姓名sname不为空的学生

```
where sname is not null
```

2.6 多个查询条件

用and（两个条件同时满足）和or（两个条件满足一个即可）

例如：查询年龄sage小于20且性别ssex为男的学生

```
where sage<20 and ssex='男'
```

2.7 分组查询（group by&聚合函数&having子句）

分组查询实现了类似excel中数据透视表的功能，可以帮助我们对数据进行分层汇总，而我们对分层后的数据进行统计的时候需要用到聚合函数，最后我们对分层之后的数据筛选的时候需要用到having子句。

**where子句是对原始表做筛选的

**having子句是对分层汇总之后的结果做筛选的

group by

group by不仅可以对一个字段进行分组，还能对多个字段进行分组。这和excel中的数据透视表一致。

2.8 聚合函数

也就是平均值、求和、最大值和最小值等

having子句

和where子句一致，只需注意是对聚合后的结果作限制。

2.9 结果呈现（order by）

和excel一样，可以用多个字段排序

关键字desc表示降序排列

例如：查询学生id和年龄，并先按照学号sid降序，再按照年龄sage升序排列

2.10 其他常用关键字

列举一些在hive取数时常用的关键字。

case when

根据字段的不同值进行不同的操作，存在大量的变形操作可以实现不同的功能，最简单的情形如下：

```
#sex字段为1和2，现在要转化为更为直观的文字形式
case sex
    when '1' then '男'
    when '2' then '女'
else '未知'
end as sex
```

count+distinct+if实现统计

```
#统计成绩单中及格同学的人数（单个学号可能出现多条记录）
count(distinct(if(score >= 60,sid,null)))
```

sum+if实现分组统计（这里sum可以替换为其他聚合函数）

```
#获取男性学生的总成绩
sum(if(sex = '男', score, 0))
```

3 SQL题，分数排名

```
SELECT  Score,(
        SELECT count(DISTINCT score)
        FROM Scores
        WHERE score >= s.score) AS Rank
FROM Scores s
ORDER BY Score DESC;
```

4 SQL题，连续三天发生购买的用户

```
SELECT * FROM ord a JOIN ord b on a.name1=b.name1 and DATEDIFF(
b.orderdate,a.orderdate)=1 JOIN ord c on c.name1=b.name1 and
DATEDIFF(c.orderdate,b.orderdate)=1;
```

5 sql中null与‘ ’的区别

null表示空，用is null判断

''表示空字符串，用=''判断

6 数据库与数据仓库的区别

简单理解下数据仓库是多个数据库以一种方式组织起来

数据库强调范式，尽可能减少冗余

数据仓库强调查询分析的速度，优化读取操作，主要目的是快速做大量数据的查询

数据仓库定期写入新数据，但不覆盖原有数据，而是给数据加上时间戳标签

数据库采用行存储，数据仓库一般采用列存储（行存储与列存储区别见题3）

数据仓库的特征是面向主题、集成、相对稳定、反映历史变化，存储数历史数据；数据库是面向事务的，存储在线交易数据

数据仓库的两个基本元素是维表和事实表，维是看待问题的角度，比如时间、部门等，事实表放着要查询的数据

7 SQL的数据类型

字符串：char、varchar、text

二进制串：binary、varbinary

布尔类型：boolean

数值类型：integer、smallint、bigint、decimal、numeric、float、real、double

时间类型：date、time、timestamp、interval

8 外键能不能为空

外键可以为空，为空表示其值还没有确定;

如果不为空，刚必须为主键相同。

9 MYSQL事务

事务是由一组SQL语句组成的逻辑处理单元，是满足 ACID 特性的一组操作，可以通过 Commit 提交一个事务，也可以使用 Rollback 进行回滚。

事务具有以下4个属性，通常简称为事务的ACID属性:

原子性（Atomicity）：事务是一个原子操作单元，其对数据的修改，要么全都执行，要么全都不执行。

一致性（Consistent）：在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数据结构（如B树索引或双向链表）也都必须是正确的。

隔离性（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。 隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

持久性（Durable）：事务完成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

10 MySQL中varchar与char的区别以及varchar(50)中的50代表的涵义

1)varchar与char的区别

char是一种固定长度的类型，varchar则是一种可变长度的类型

2)varchar(50)中50的涵义

最多存放50个字符，varchar(50)和(200)存储hello所占空间一样，但后者在排序时会消耗更多内存，因为order by col采用fixed_length计算col长度(memory引擎也一样)

3)int（20）中20的涵义

是指显示字符的长度

但要加参数的，最大为255，比如它是记录行数的id,插入10笔资料，它就显示00000000001 ~~~00000000010，当字符的位数超过11,它也只显示11位，如果你没有加那个让它未满11位就前面加0的参数，它不会在前面加0。

20表示最大显示宽度为20，但仍占4字节存储，存储范围不变。

4)mysql为什么这么设计

对大多数应用没有意义，只是规定一些工具用来显示字符的个数；int(1)和int(20)存储和计算均一样；

11 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NULL

key_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

12 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;
```

13 索引是什么？有什么作用以及优缺点?

1）索引是对数据库表中一或多个列的值进行排序的结构，是帮助MySQL高效获取数据的数据结构

2）索引就是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

14 MySQL数据库几个基本的索引类型：普通索引、唯一索引、主键索引、全文索引

1）索引加快数据库的检索速度

2）索引降低了插入、删除、修改等维护任务的速度

3）唯一索引可以确保每一行数据的唯一性

4）通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能

5）索引需要占物理和数据空间

15 使用索引查询一定能提高查询的性能吗？为什么?

通常，通过索引查询数据比全表扫描要快。但是我们也必须注意到它的代价。

1）索引需要空间来存储，也需要定期维护，每当有记录在表中增减或索引列被修改时，索引本身也会被修改。这意味着每条记录的INSERT，DELETE，UPDATE将为此多付出4,5 次的磁盘I/O。因为索引需要额外的存储空间和处理，那些不必要的索引反而会使查询反应时间变慢。使用索引查询不一定能提高查询性能，索引范围查询(INDEX RANGE SCAN)适用于两种情况。

2）基于一个范围的检索,一般查询返回结果集小于表中记录数的30%

3）基于非唯一性索引的检索

16 简单说一说drop、delete与truncate的区别

SQL中的drop、delete、truncate都表示删除，但是三者有一些差别

1）delete和truncate只删除表的数据不删除表的结构

2）速度，一般来说: drop> truncate >delete

3）delete语句是dml，这个操作会放到rollback segement中，事务提交之后才生效;

4）如果有相应的trigger，执行的时候将被触发。truncate，drop是ddl，，操作立即生效，原数据不放到rollback segment中，不能回滚。 操作不触发trigger。

17 drop、delete与truncate分别在什么场景之下使用?

1）不再需要一张表的时候，用drop

2）想删除部分数据行时候，用delete，并且带上where子句

3）保留表而删除所有数据的时候用truncate

18 什么是视图？以及视图的使用场景有哪些?

1）视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

2）只暴露部分字段给访问者，所以就建一个虚表，就是视图。

3）查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异

19 对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

20 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

21 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num is null
```

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

```
select id from t where num=0
```

22 应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num=10 or num=20
```

可以这样查询：

```
select id from t where num=10
union all
select id from t where num=20
```

23 下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

若要提高效率，可以考虑全文检索。

24 in和 not in 也要慎用，否则会导致全表扫描，如：

```
select id from t where num in(1,2,3)
```

对于连续的数值，能用 between 就不要用 in 了：

```
select id from t where num between 1 and 3
```

25 如果在 where 子句中使用参数，也会导致全表扫描。

如下面语句将进行全表扫描：

```
select id from t where num=@num
```

可以改为强制查询使用索引：

```
select id from t with(index(索引名)) where num=@num
```

26 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where num/2=100
```

应改为:

```
select id from t where num=100*2
```

27 应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

```
select id from t where substring(name,1,3)='abc'
```

--name以abc开头的id

```
select id from t where datediff(day,createdate,'2005-11-30')=0
```

--'2005-11-30'生成的id

应改为:

```
select id from t where name like 'abc%'
select id from t where createdate>='2005-11-30' and createdate<'2005-12-1'
```

28 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将不能正确使用索引。

29 在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

30 不要写一些没有意义的查询，如需要生成一个空表结构：

```
select col1,col2 into #t from t where 1=0
```

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：

```
create table #t(...)
```

31 很多时候用 exists 代替 in 是一个好的选择：

```
select num from a where num in(select num from b)
```

用下面的语句替换：

```
select num from a where exists(select 1 from b where num=a.num)
```

32 并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

33 索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

34 应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

35 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

36 尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

37 任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

38 尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

39 避免频繁创建和删除临时表，以减少系统表资源的消耗。

40 临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

41 在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

42 如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

43 尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

44 使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

45 尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

46 尽量避免大事务操作，提高系统并发能力。

47 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.

48 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.

49 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。

50 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。

51 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。

52 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?

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。

53 索引概念、索引模型（阿里完整的面试题以及答案，需要可以细看）。（这个很有价值，一般用不上有需要的可以细看。）

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：

你们每天这么大的数据量，都是保存在关系型数据库中吗？

是的，我们线上使用的是MySQL数据库

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？

我们在数据库中创建了一些索引。

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。

那你能说说什么是索引吗？

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。

那么索引具体采用的哪种数据结构呢？

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。

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？

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。

除了上面这个范围查询的，你还能说出其他的一些区别吗？

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

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？

InnoDB的B+ Tree可能存储的是整行数据，也有可能是主键的值。

那这两者有什么区别吗？

（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。

那么，聚簇索引和非聚簇索引，在查询数据的时候有区别吗？

聚簇索引查询会更快？

为什么呢？

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。

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？

（通过覆盖索引也可以只查询一次）

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

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

那你们有用过联合索引吗?

用过呀，我们有对一些表中创建过联合索引。

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?

我们把识别度最高的字段放到最前面。

为什么这么做呢?

那你知道最左前缀匹配吗?

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

你们线上用的MySQL是哪个版本啊呢?

我们MySQL是5.7

那你知道在MySQL 5.6中，对索引做了哪些优化吗?

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）

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。

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?

这个还没有统计过，除非遇到慢SQL的时候我们才会去排查

那排查的时候，有什么手段可以知道有没有走索引查询呢?

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

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?

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。