1. 索引是什么?

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

索引是一种数据结构。数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B树及其变种B+树。更通俗的说，索引就相当于目录。为了方便查找书中的内容，通过对内容建立索引形成目录。而且索引是一个文件，它是要占据物理空间的。

MySQL索引的建立对于MySQL的高效运行是很重要的，索引可以大大提高MySQL的检索速度。比如我们在查字典的时候，前面都有检索的拼音和偏旁、笔画等，然后找到对应字典页码，这样然后就打开字典的页数就可以知道我们要搜索的某一个key的全部值的信息了。

2. 索引有哪些优缺点?

索引的优点

- 可以大大加快数据的检索速度，这也是创建索引的最主要的原因。
- 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。

索引的缺点

- 时间方面：创建索引和维护索引要耗费时间，具体地，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，会降低增/改/删的执行效率；
- 空间方面：索引需要占物理空间。

3. MySQL有哪几种索引类型?

1、从存储结构上来划分：BTree索引（B-Tree或B+Tree索引），Hash索引，full-index全文索引，R-Tree索引。这里所描述的是索引存储时保存的形式，

2、从应用层次来分：普通索引，唯一索引，复合索引。

- 普通索引：即一个索引只包含单个列，一个表可以有多个单列索引
- 唯一索引：索引列的值必须唯一，但允许有空值
- 复合索引：多列值组成一个索引，专门用于组合搜索，其效率大于索引合并
- 聚簇索引(聚集索引)：并不是一种单独的索引类型，而是一种数据存储方式。具体细节取决于不同的实现，InnoDB的聚簇索引其实就是在同一个结构中保存了B-Tree索引(技术上来说是B+Tree)和数据行。
- 非聚簇索引： 不是聚簇索引，就是非聚簇索引

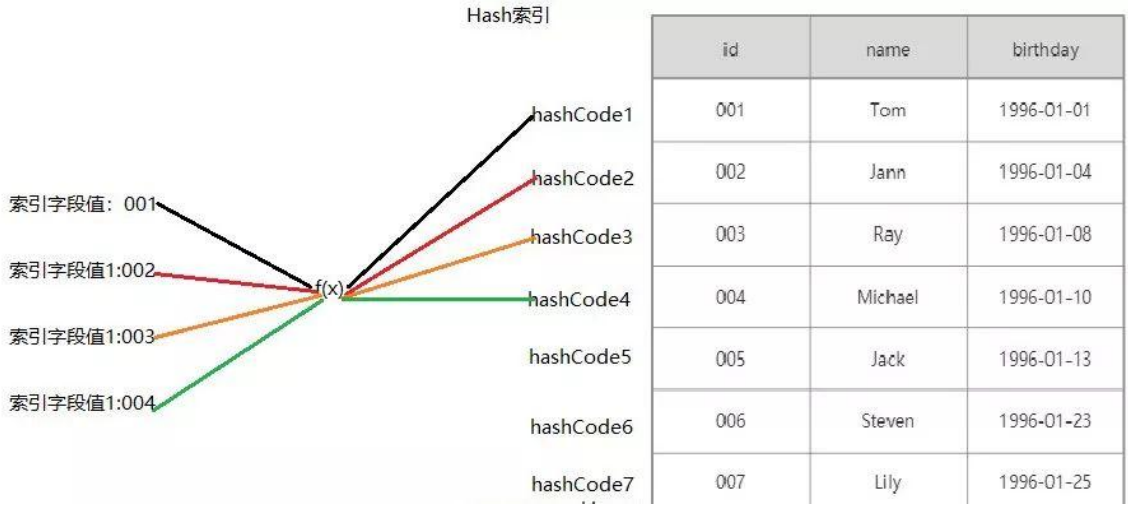
3、根据中数据的物理顺序与键值的逻辑（索引）顺序关系： 聚集索引，非聚集索引。

4. 说一说索引的底层实现?

Hash索引

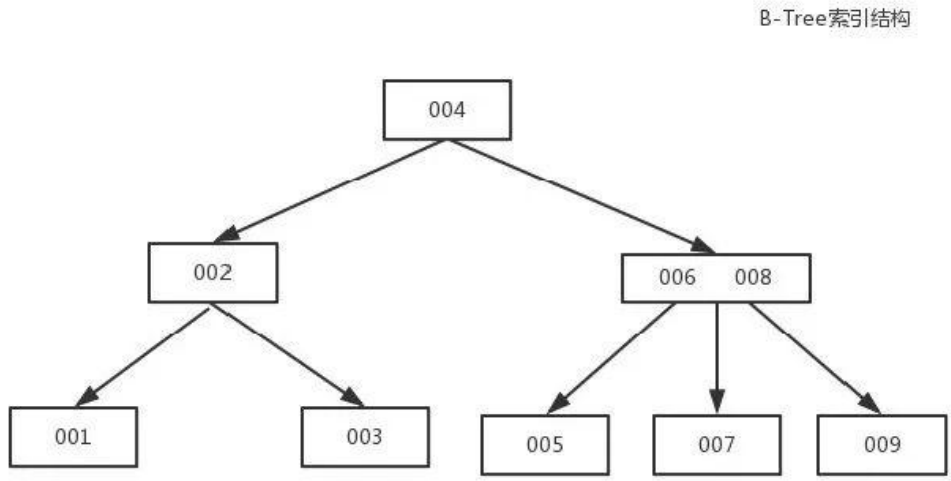
基于哈希表实现，只有精确匹配索引所有列的查询才有效，对于每一行数据，存储引擎都会对所有的索引列计算一个哈希码（hash code），并且Hash索引将所有的哈希码存储在索引中，同时在索引表中保存指向每个数据行的指针。

> 图片来源：https://www.javazhiyin.com/40232.html



B-Tree索引（MySQL使用B+Tree）

B-Tree能加快数据的访问速度，因为存储引擎不再需要进行全表扫描来获取数据，数据分布在各个节点之中。

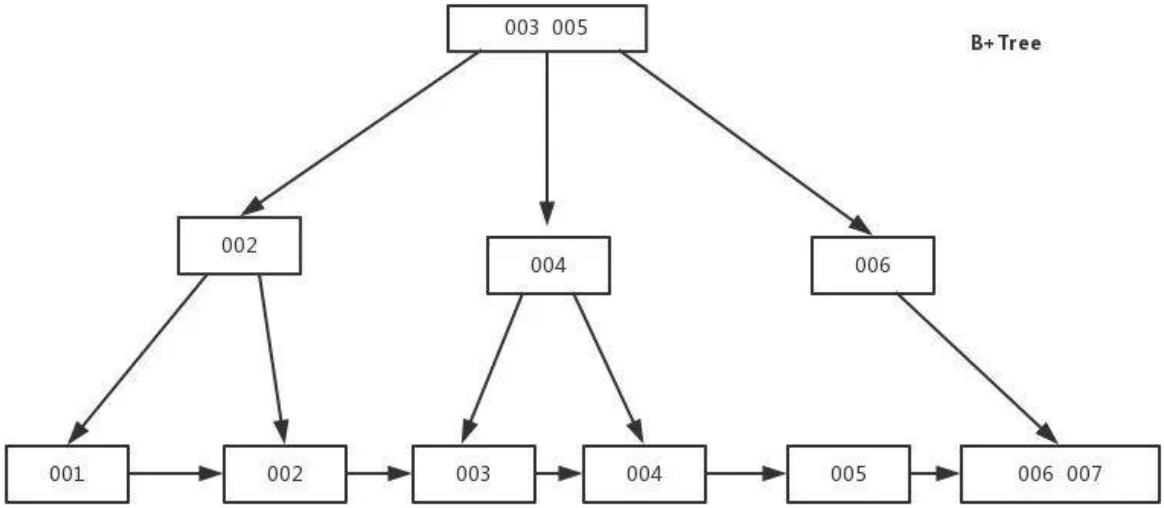


B+Tree索引

是B-Tree的改进版本，同时也是数据库索引索引所采用的存储结构。数据都在叶子节点上，并且增加了顺序访问指针，每个叶子节点都指向相邻的叶子节点的地址。相比B-Tree来说，进行范围查找时只需要查找两个节点，进行遍历即可。而B-Tree需要获取所有节点，相比之下B+Tree效率更高。

B+tree性质：

- n棵子tree的节点包含n个关键字，不用来保存数据而是保存数据的索引。
- 所有的叶子结点中包含了全部关键字的信息，及指向含这些关键字记录的指针，且叶子结点本身依关键字的大小自小而大顺序链接。
- 所有的非终端结点可以看成是索引部分，结点中仅含其子树中的最大（或最小）关键字。
- B+ 树中，数据对象的插入和删除仅在叶节点上进行。
- B+树有2个头指针，一个是树的根节点，一个是最小关键码的叶节点。



5. 为什么索引结构默认使用B+Tree，而不是B-Tree，Hash，二叉树，红黑树?

B-tree： 从两个方面来回答

- B+树的磁盘读写代价更低：B+树的内部节点并没有指向关键字具体信息的指针，因此其内部节点相对B(B-)树更小，如果把所有同一内部节点的关键字存放在同一盘块中，那么盘块所能容纳的关键字数量也越多，一次性读入内存的需要查找的关键字也就越多，相对`IO读写次数就降低`了。
- 由于B+树的数据都存储在叶子结点中，分支结点均为索引，方便扫库，只需要扫一遍叶子结点即可，但是B树因为其分支结点同样存储着数据，我们要找到具体的数据，需要进行一次中序遍历按序来扫，所以B+树更加适合在`区间查询`的情况，所以通常B+树用于数据库索引。

Hash：

- 虽然可以快速定位，但是没有顺序，IO复杂度高；
- 基于Hash表实现，只有Memory存储引擎显式支持哈希索引；
- 适合**等值查询**，如=、in()、<=>，不支持范围查询；
- 因为不是按照索引值顺序存储的，就不能像B+Tree索引一样利用索引完成排序；
- Hash索引在查询等值时非常快；
- 因为Hash索引始终索引的**所有列的全部内容**，所以不支持部分索引列的匹配查找；
- 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 。

二叉树： 树的高度不均匀，不能自平衡，查找效率跟数据有关（树的高度），并且IO代价高。

红黑树： 树的高度随着数据量增加而增加，IO代价高。

6. 讲一讲聚簇索引与非聚簇索引?

都是B+树的数据结构

- 聚簇索引：将数据存储和索引放到了一起，并且是按照一定的顺序组织的，找到索引就找到了数据，数据的物理存放顺序与索引的顺序是一致的，即：只要索引是相邻的，那么对应的数据一定也是相邻的存放在磁盘上。
- 非聚簇索引：叶子节点不存储数据，存储的是数据行的地址，也就是说根据索引查找到数据行的位置再取磁盘查找数据，即回表。

聚簇索引与非聚簇索引的区别：

- 非聚集索引与聚集索引的区别在于非聚集索引的叶子节点不存储表中的数据，而是存储该记录对应的主键（行号）
- 对于InnoDB来说，想要查找数据我们还需要根据主键再去聚集索引中进行查找，这个再根据聚集索引查找数据的过程，我们称为**回表**。第一次索引一般是顺序IO，回表的操作属于随机IO。需要回表的次数越多，即随机IO次数越多，我们就越倾向于使用全表扫描 。
- 通常情况下， 主键索引（聚簇索引）查询只会查一次，而非主键索引（非聚簇索引）需要回表查询多次。当然，如果是覆盖索引的话，查一次即可

> 注意：**MyISAM无论主键索引还是二级索引都是非聚簇索引，而InnoDB的主键索引是聚簇索引，二级索引是非聚簇索引。我们自己建的索引基本都是非聚簇索引。**

优势：
1、查询通过聚簇索引可以直接获取数据，相比非聚簇索引需要第二次查询（非覆盖索引的情况下）效率要高
2、聚簇索引对于范围查询的效率很高，因为其数据是按照大小排列的
3、聚簇索引适合用在排序的场合，非聚簇索引不适合

劣势：
1、维护索引很昂贵，特别是插入新行或者主键被更新导至要分页(page split)的时候。建议在大量插入新行后，选在负载较低的时间段，通过OPTIMIZE TABLE优化表，因为必须被移动的行数据可能造成碎片。使用独享表空间可以弱化碎片
2、表因为使用UUId（随机ID）作为主键，使数据存储稀疏，这就会出现聚簇索引有可能有比全表扫面更慢，所以建议使用int的auto_increment作为主键
3、如果主键比较大的话，那辅助索引将会变的更大，因为辅助索引的叶子存储的是主键值；过长的主键值，会导致非叶子节点占用占用更多的物理空间

InnoDB中一定有主键，主键一定是聚簇索引，不手动设置、则会使用unique索引，没有unique索引，则会使用数据库内部的一个行的隐藏id来当作主键索引。在聚簇索引之上创建的索引称之为辅助索引，辅助索引访问数据总

InnoDB中一定有主键，主键一定是聚簇索引，不手动设置、则会使用unique索引，没有unique索引，则会使用数据库内部的一个行的隐藏id来当作主键索引。在聚簇索引之上创建的索引称之为辅助索引，辅助索引访问数据总是需要二次查找，非聚簇索引都是辅助索引，像复合索引、前缀索引、唯一索引，辅助索引叶子节点存储的不再是行的物理位置，而是主键值

MyISM使用的是非聚簇索引，没有聚簇索引，非聚簇索引的两棵B+树看上去没什么不同，节点的结构完全一致只是存储的内容不同而已，主键索引B+树的节点存储了主键，辅助键索引B+树存储了辅助键。表数据存储在独立的地方，这两颗B+树的叶子节点都使用一个地址指向真正的表数据，对于表数据来说，这两个键没有任何差别。由于索引树是独立的，通过辅助键检索无需访问主键的索引树。

如果涉及到大数据量的排序、全表扫描、count之类的操作的话，还是MyISAM占优势些，因为索引所占空间小，这些操作是需要在内存中完成的。

7. 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。一个索引包含（覆盖）所有需要查询字段的值，被称之为"覆盖索引"。

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 score from student where score > 90`的查询时，在索引的叶子节点上，已经包含了score 信息，不会再次进行回表查询。

8. 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。

9. 讲一讲MySQL的最左前缀原则?

当一个SQL想要使用索引时，就一定要提供该索引对应的字段中最左边的字段，也就是排在最前面的字段。最左前缀原则就是最左优先，在创建多列索引（联合索引）时，要根据业务需求，where子句中使用最频繁的一列放在最左边。

mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。

=和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式。

> 在MySQL建立联合索引时会遵守最左前缀匹配原则，即最左优先，在检索数据时从联合索引的最左边开始匹配。

10. 讲一讲前缀索引?

因为可能我们索引的字段非常长，这既占内存空间，也不利于维护。所以我们就想，如果只把很长字段的前面的公共部分作为一个索引，就会产生超级加倍的效果。但是，我们需要注意，order by不支持前缀索引 。

流程是:

先计算完整列的选择性 : `select count(distinct col_1)/count(1) from table_1`

再计算不同前缀长度的选择性 : `select count(distinct left(col_1,4))/count(1) from table_1`

找到最优长度之后，创建前缀索引 : `create index idx_front on table_1 (col_1(4))`

11. 了解索引下推吗?

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。

- 有了索引下推优化，可以在**减少回表次数**
- 在InnoDB中只针对二级索引有效

官方文档中给的例子和解释如下:

在 people_table中有一个二级索引(zipcode，lastname，address)，查询是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%' and 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%' and 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。

12. 怎么查看MySQL语句有没有用到索引?

通过explain，如以下例子：

```
EXPLAIN SELECT * FROM employees.titles WHERE emp_no='10001' AND title='Senior Engineer' AND from_date='1986-06-26';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	filtered	rows	Extra
1	SIMPLE	titles	null	const	PRIMARY	PRIMARY	59	const,const,const	10	1	

- id：在一个大的查询语句中每个**SELECT**关键字都对应一个唯一的id，如explain select * from s1 where id = (select id from s1 where name = 'egon1');第一个select的id是1，第二个select的id是2。有时候会出现两个select，但是id却都是1，这是因为优化器把子查询变成了连接查询。
- select_type：select关键字对应的那个查询的类型，如SIMPLE,PRIMARY,SUBQUERY,DEPENDENT,SNION。
- table：每个查询对应的表名。
- type：`type` 字段比较重要, 它提供了判断查询是否高效的重要依据依据. 通过 `type` 字段, 我们判断此次查询是 `全表扫描` 还是 `索引扫描` 等。如const(主键索引或者唯一二级索引进行等值匹配的情况下),ref(普通的二级索引列与常量进行等值匹配),index(扫描全表索引的覆盖索引)。

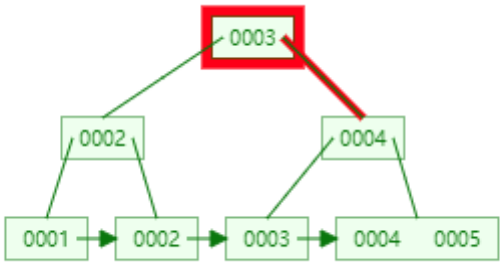
  通常来说, 不同的 type 类型的性能关系如下:
  `ALL < index < range ~ index_merge < ref < eq_ref < const < system`
  `ALL` 类型因为是全表扫描, 因此在相同的查询条件下, 它是速度最慢的.
  而 `index` 类型的查询虽然不是全表扫描, 但是它扫描了所有的索引, 因此比 ALL 类型的稍快.
- possible_key：查询中可能用到的索引(可以把用不到的删掉，降低优化器的优化时间)。
- key：此字段是 MySQL 在当前查询时所真正使用到的索引。
- filtered：查询器预测满足下一次查询条件的百分比。
- rows 也是一个重要的字段. MySQL 查询优化器根据统计信息, 估算 SQL 要查找到结果集需要扫描读取的数据行数.
  这个值非常直观显示 SQL 的效率好坏, 原则上 rows 越少越好。
- extra：表示额外信息，如Using where,Start temporary,End temporary,Using temporary等。

13. 为什么官方建议使用自增长主键作为索引?

结合B+Tree的特点，自增主键是连续的，在插入过程中尽量减少页分裂，即使要进行页分裂，也只会分裂很少一部分。并且能减少数据的移动，每次插入都是插入到最后。总之就是减少分裂和移动的频率。

插入连续的数据：

> 图片来自：https://www.javazhiyin.com/40232.html



插入非连续的数据：

14. 如何创建索引?

创建索引有三种方式。

1、 在执行CREATE TABLE时创建索引

```
CREATE TABLE user_index2 (
    id INT auto_increment PRIMARY KEY,
    first_name VARCHAR (16),
    last_name VARCHAR (16),
    id_card VARCHAR (18),
    information text,
    KEY name (first_name, last_name),
    FULLTEXT KEY (information),
    UNIQUE KEY (id_card)
);
```

2、 使用ALTER TABLE命令去增加索引。

```
ALTER TABLE table_name ADD INDEX index_name (column_list);
```

ALTER TABLE用来创建普通索引、UNIQUE索引或PRIMARY KEY索引。

其中table_name是要增加索引的表名，column_list指出对哪些列进行索引，多列时各列之间用逗号分隔。

索引名index_name可自己命名，缺省时，MySQL将根据第一个索引列赋一个名称。另外，ALTER TABLE允许在单个语句中更改多个表，因此可以在同时创建多个索引。
3、 使用CREATE INDEX命令创建。

```
CREATE INDEX index_name ON table_name (column_list);
```

15. 创建索引时需要注意什么?

- 非空字段：应该指定列为NOT NULL，除非你想存储NULL。在mysql中，含有空值的列很难进行查询优化，因为它们使得索引、索引的统计信息以及比较运算更加复杂。你应该用0、一个特殊的值或者一个空串代替空值；
- 取值离散大的字段：（变量各个取值之间的差异程度）的列放到联合索引的前面，可以通过count()函数查看字段的差异值，返回值越大说明字段的唯一值越多字段的离散程度高；
- 索引字段越小越好：数据库的数据存储以页为单位一页存储的数据越多一次IO操作获取的数据越大效率越高。

16. 建索引的原则有哪些?

1、最左前缀匹配原则，非常重要的原则，mysql会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配，比如a = 1 and b = 2 and c > 3 and d = 4 如果建立(a,b,c,d)顺序的索引，d是用不到索引的，如果建立(a,b,d,c)的索引则都可以用到，a,b,d的顺序可以任意调整。

2、=和in可以乱序，比如a = 1 and b = 2 and c = 3 建立(a,b,c)索引可以任意顺序，mysql的查询优化器会帮你优化成索引可以识别的形式。

3、尽量选择区分度高的列作为索引，区分度的公式是count(distinct col)/count(*)，表示字段不重复的比例，比例越大我们扫描的记录数越少，唯一键的区分度是1，而一些状态、性别字段可能在大数据面前区分度就是0，那可能有人会问，这个比例有什么经验值吗？使用场景不同，这个值也很难确定，一般需要join的字段我们都要求是0.1以上，即平均1条扫描10条记录。

4、索引列不能参与计算，保持列“干净”，比如from_unixtime(create_time) = ’2014-05-29’就不能使用到索引，原因很简单，b+树中存的都是数据表中的字段值，但进行检索时，需要把所有元素都应用函数才能比较，显然成本太大。所以语句应该写成create_time = unix_timestamp(’2014-05-29’)。

5、尽量的扩展索引，不要新建索引。比如表中已经有a的索引，现在要加(a,b)的索引，那么只需要修改原来的索引即可。

17. 使用索引查询一定能提高查询的性能吗?

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。

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I* NSERT，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:

- 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%。
- 基于非唯一性索引的检索。

18. 什么情况下不走索引（索引失效）？

1、使用!= 或者 <> 导致索引失效

2、类型不一致导致的索引失效

3、函数导致的索引失效

如:

```
SELECT * FROM `user` WHERE DATE(create_time) = '2020-09-03';
```

如果使用函数在索引列，这是不走索引的。

4、运算符导致的索引失效

```
SELECT * FROM `user` WHERE age - 1 = 20;
```

如果你对列进行了（+，-，*，/，!），那么都将不会走索引。

5、OR引起的索引失效

```
SELECT * FROM `user` WHERE `name` = '张三' OR height = '175';
```

OR导致索引是在特定情况下的，并不是所有的OR都是使索引失效，如果OR连接的是同一个字段，那么索引不会失效，反之索引失效。

6、模糊搜索导致的索引失效

```
SELECT * FROM `user` WHERE `name` LIKE '%冰';
```

当`%`放在匹配字段前是不走索引的，放在后面才会走索引。

7、NOT IN、NOT EXISTS导致索引失效