聊聊sql优化的15个小技巧

捡田螺的小男孩 2021-11-24 08:14

以下文章来源于苏三说技术 ， 作者因为热爱所以坚持ing

苏三说技术

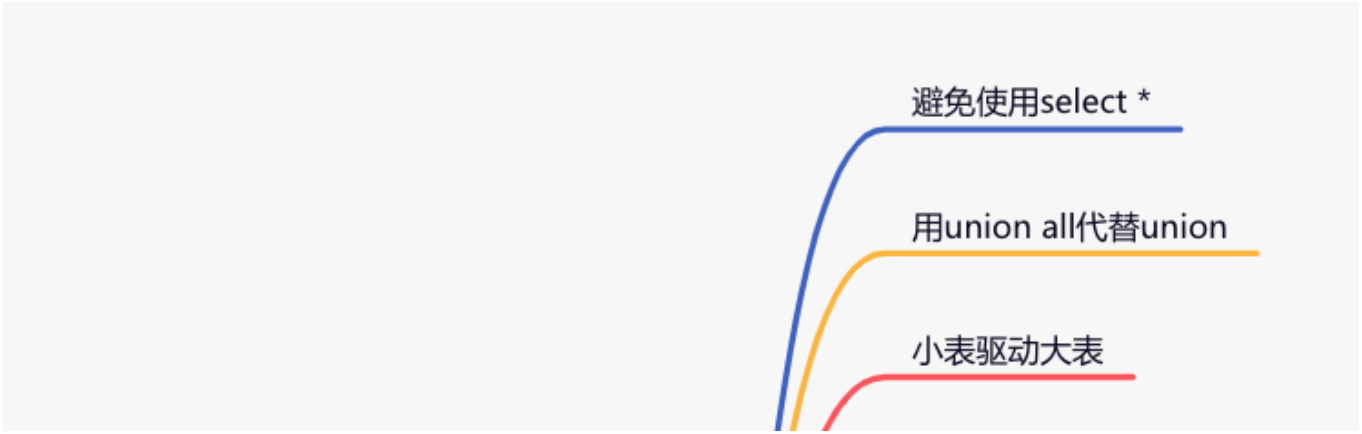
作者就职于知名互联网公司，掘金月度优秀作者，从事开发、架构和部分管理工作。实战经验丰富，对jdk、spring、springboot、sprin...

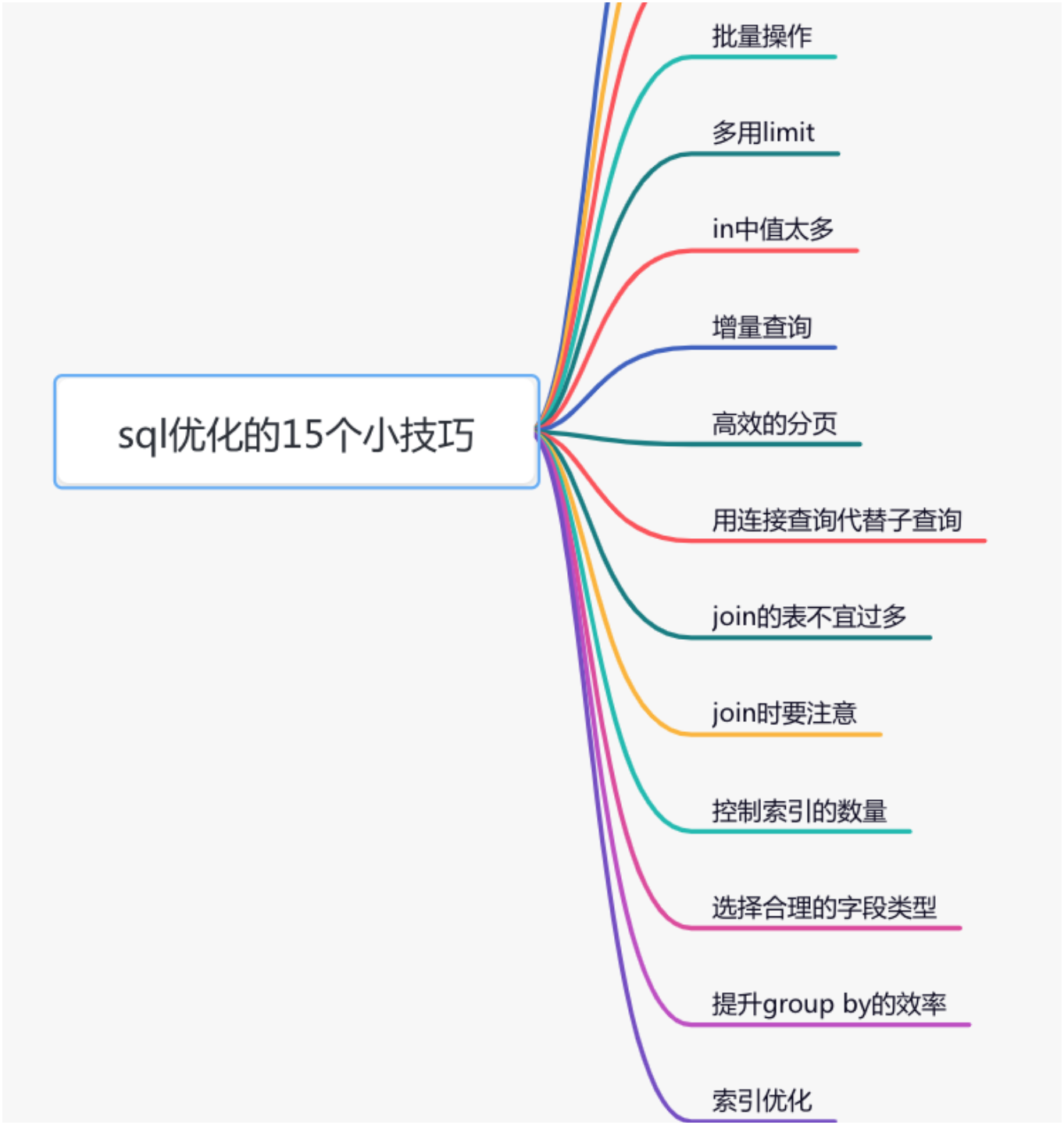
sql优化是一个大家都比较关注的热门话题，无论你在面试，还是工作中，都很有可能会遇到。

如果某天你负责的某个线上接口，出现了性能问题，需要做优化。那么你首先想到的很有可能是优化sql语句，因为它的改造成本相对于代码来说也要小得多。

那么，如何优化sql语句呢？

这篇文章从15个方面，分享了sql优化的一些小技巧，希望对你有所帮助。




sql优化的15个小技巧
批量操作
多用limit
in中值太多
增量查询
高效的分页
用连接查询代替子查询
join的表不宜过多
join时要注意
控制索引的数量
选择合理的字段类型
提升group by的效率
索引优化

1 避免使用select *

很多时候，我们写sql语句时，为了方便，喜欢直接使用 `select *` ，一次性查出表中所有列的数据。

反例：

```
select * from user where id=1;
```

在实际业务场景中，可能我们真正需要使用的只有其中一两列。查了很多数据，但是不用，白白浪费了数据库资源，比如：内存或者cpu。

此外，多查出来的数据，通过网络IO传输的过程中，也会增加数据传输的时间。

还有一个最重要的问题是： `select *` 不会走 覆盖索引 ，会出现大量的 回表 操作，而从导致查询sql的性能很低。

那么，如何优化呢？

正例：

```
select name,age from user where id=1;
```

```
select name,age from user where id=1;
```

sql语句查询时，只查需要用到的列，多余的列根本无需查出来。

2 用union all代替union

我们都知道sql语句使用 `union` 关键字后，可以获取排重后的数据。

而如果使用 `union all` 关键字，可以获取所有数据，包含重复的数据。

反例：

```
(select * from user where id=1)
union
(select * from user where id=2);
```

排重的过程需要遍历、排序和比较，它更耗时，更消耗cpu资源。

所以如果能用union all的时候，尽量不用union。

正例：

```
(select * from user where id=1)
union all
(select * from user where id=2);
```

除非是有些特殊的场景，比如union all之后，结果集中出现了重复数据，而业务场景中是不允许产生重复数据的，这时可以使用union。

3 小表驱动大表

小表驱动大表，也就是说用小表的数据集驱动大表的数据集。

假如有order和user两张表，其中order表有10000条数据，而user表有100条数据。

这时如果想查一下，所有有效的用户下过的订单列表。

可以使用 `in` 关键字实现：

```
select * from order
where user_id in (select id from user where status=1)
```

也可以使用 `exists` 关键字实现：

```
select * from order
where exists (select 1 from user where order.user_id = user.id and status=1)
```

前面提到的这种业务场景，使用in关键字去实现业务需求，更加合适。

为什么呢？

因为如果sql语句中包含了in关键字，则它会优先执行in里面的 子查询语句 ，然后再执行in外面的语句。如果in里面的数据量很少，作为条件查询速度更快。

而如果sql语句中包含了exists关键字，它优先执行exists左边的语句（即主查询语句）。然后把它作为条件，去跟右边的语句匹配。如果匹配上，则可以查询出数据。如果匹配不上，数据就被过滤掉了。

这个需求中，order表有10000条数据，而user表有100条数据。order表是大表，user表是小表。如果order表在左边，则用in关键字性能更好。

总结一下：

- in 适用于左边大表，右边小表。
- exists 适用于左边小表，右边大表。

不管是用in，还是exists关键字，其核心思想都是用小表驱动大表。

4 批量操作

如果你有一批数据经过业务处理之后，需要插入数据，该怎么办？

反例：

```
for(Order order: list){
   orderMapper.insert(order):
}
```

在循环中逐条插入数据。

```
insert into order(id,code,user_id)
values(123,'001',100);
```

该操作需要多次请求数据库，才能完成这批数据的插入。

但众所周知，我们在代码中，每次远程请求数据库，是会消耗一定性能的。而如果我们的代码需要请求多次数据库，才能完成本次业务功能，势必会消耗更多的性能。

那么如何优化呢？

正例：

```
orderMapper.insertBatch(list):
```

提供一个批量插入数据的方法。

```
insert into order(id,code,user_id)
values(123,'001',100),(124,'002',100),(125,'003',101);
```

这样只需要远程请求一次数据库，sql性能会得到提升，数据量越多，提升越大。

但需要注意的是，不建议一次批量操作太多的数据，如果数据太多数据库响应也会很慢。批量操作需要把握一个度，建议每批数据尽量控制在500以内。如果数据多于500，则分多批次处理。

5 多用limit

有时候，我们需要查询某些数据中的第一条，比如：查询某个用户下的第一个订单，想看看他第一次的首单时间。

反例：

```
select id, create_date
 from order
where user_id=123
order by create_date asc;
```

根据用户id查询订单，按下单时间排序，先查出该用户所有的订单数据，得到一个订单集合。然后在代码中，获取第一个元素的数据，即首单的数据，就能获取首单时间。

```
List<Order> list = orderMapper.getOrderList();
Order order = list.get(0);
```

虽说这种做法在功能上没有问题，但它的效率非常不高，需要先查询出所有的数据，有点浪费资源。

那么，如何优化呢？

正例：

```
select id, create_date
 from order
where user_id=123
order by create_date asc
limit 1;
```

使用 `limit 1` ，只返回该用户下单时间最小的那一条数据即可。

> 此外，在删除或者修改数据时，为了防止误操作，导致删除或修改了不相干的数据，也可以在sql语句最后加上limit。

例如：

```
update order set status=0,edit_time=now(3)
where id>=100 and id<200 limit 100;
```

这样即使误操作，比如把id搞错了，也不会对太多的数据造成影响。

6 in中值太多

对于批量查询接口，我们通常会使用 `in` 关键字过滤出数据。比如：想通过指定的一些id，批量查询出用户信息。

sql语句如下：

```
select id,name from category
where id in (1,2,3...100000000);
```

如果我们不做任何限制，该查询语句一次性可能会查询出非常多的数据，很容易导致接口超时。

这时该怎么办呢？

```
select id,name from category
where id in (1,2,3...100)
limit 500;
```

可以在sql中对数据用limit做限制。

不过我们更多的是要在业务代码中加限制，伪代码如下：

```
public List<Category> getCategory(List<Long> ids) {
   if(CollectionUtils.isEmpty(ids)) {
      return null;
   }
   if(ids.size() > 500) {
      throw new BusinessException("一次最多允许查询500条记录")
   }
   return mapper.getCategoryList(ids);
}
```

还有一个方案就是：如果ids超过500条记录，可以分批用多线程去查询数据。每批只查500条记录，最后把查询到的数据汇总到一起返回。

不过这只是一个临时方案，不适合于ids实在太多的场景。因为ids太多，即使能快速查出数据，但如果返回的数据量太大了，网络传输也是非常消耗性能的，接口性能始终好不到哪里去。

7 增量查询

有时候，我们需要通过远程接口查询数据，然后同步到另外一个数据库。

反例：

```
select * from user;
```

如果直接获取所有的数据，然后同步过去。这样虽说非常方便，但是带来了一个非常大的问题，就是如果数据很多的话，查询性能会非常差。

这时该怎么办呢？

正例：

```
select * from user
where id>#{lastId} and create_time >= #{lastCreateTime}
limit 100;
```

按id和时间升序，每次只同步一批数据，这一批数据只有100条记录。每次同步完成之后，保存这100条数据中最大的id和时间，给同步下一批数据的时候用。

通过这种增量查询的方式，能够提升单次查询的效率。

8 高效的分页

有时候，列表页在查询数据时，为了避免一次性返回过多的数据影响接口性能，我们一般会对查询接口做分页处理。

在mysql中分页一般用的 `limit` 关键字：

```
select id,name,age
from user limit 10,20;
```

如果表中数据量少，用limit关键字做分页，没啥问题。但如果表中数据量很多，用它就会出现性能问题。

比如现在分页参数变成了：

```
select id,name,age
from user limit 1000000,20;
```

mysql会查到1000020条数据，然后丢弃前面的1000000条，只查后面的20条数据，这个是非常浪费资源的。

那么，这种海量数据该怎么分页呢？

优化sql：

```
select id,name,age
from user where id > 1000000 limit 20;
```

先找到上次分页最大的id，然后利用id上的索引查询。不过该方案，要求id是连续的，并且有序的。

还能使用 `between` 优化分页。

```
select id,name,age
from user where id between 1000000 and 1000020;
```

需要注意的是between要在唯一索引上分页，不然会出现每页大小不一致的问题。

9 用连接查询代替子查询

mysql中如果需要从两张以上的表中查询出数据的话，一般有两种实现方式： 子查询 和 连接查询 。

子查询的例子如下：

```
select * from order
where user_id in (select id from user where status=1)
```

子查询语句可以通过 in 关键字实现，一个查询语句的条件落在另一个select语句的查询结果中。程序先运行在嵌套在最内层的语句，再运行外层的语句。

子查询语句的优点是简单，结构化，如果涉及的表数量不多的话。

但缺点是mysql执行子查询时，需要创建临时表，查询完毕后，需要再删除这些临时表，有一些额外的性能消耗。

这时可以改成连接查询。具体例子如下：

```
select o.* from order o
inner join user u on o.user_id = u.id
where u.status=1
```

10 join的表不宜过多

根据阿里巴巴开发者手册的规定，join表的数量不应该超过 3 个。

反例：

```
select a.name,b.name.c.name,d.name
from a
inner join b on a.id = b.a_id
inner join c on c.b_id = b.id
inner join d on d.c_id = c.id
inner join e on e.d_id = d.id
inner join f on f.e_id = e.id
inner join g on g.f_id = f.id
```

如果join太多，mysql在选择索引的时候会非常复杂，很容易选错索引。

并且如果没有命中中，nested loop join 就是分别从两个表读一行数据进行两两对比，复杂度是 n^2。

所以我们应该尽量控制join表的数量。

正例：

```
select a.name,b.name.c.name,a.d_name
from a
inner join b on a.id = b.a_id
inner join c on c.b_id = b.id
```

如果实现业务场景中需要查询出另外几张表中的数据，可以在a、b、c表中 冗余专门的字段 ，比如：在表a中冗余d_name字段，保存需要查询出的数据。

不过我之前也见过有些ERP系统，并发量不大，但业务比较复杂，需要join十几张表才能查询出数据。

所以join表的数量要根据系统的实际情况决定，不能一概而论，尽量越少越好。

11 join时要注意

我们在涉及到多张表联合查询的时候，一般会使用 `join` 关键字。

而join使用最多的是left join和inner join。

- `left join` ：求两个表的交集外加左表剩下的数据。
- `inner join` ：求两个表交集的数据。

使用inner join的示例如下：

```
select o.id,o.code,u.name
from order o
inner join user u on o.user_id = u.id
where u.status=1;
```

如果两张表使用inner join关联，mysql会自动选择两张表中的小表，去驱动大表，所以性能上不会有太大的问题。

使用left join的示例如下：

```
select o.id,o.code,u.name
from order o
left join user u on o.user_id = u.id
where u.status=1;
```

如果两张表使用left join关联，mysql会默认用left join关键字左边的表，去驱动它右边的表。如果左边的表数据很多时，就会出现性能问题。

> 要特别注意的是在用left join关联查询时，左边要用小表，右边可以用大表。如果能用inner join的地方，尽量少用left join。

12 控制索引的数量

众所周知，索引能够显著的提升查询sql的性能，但索引数量并非越多越好。

因为表中新增数据时，需要同时为它创建索引，而索引是需要额外的存储空间的，而且还会有一定的性能消耗。

阿里巴巴的开发者手册中规定，单表的索引数量应该尽量控制在 5 个以内，并且单个索引中的字段数不超过 5 个。

mysql使用的B+树的结构来保存索引的，在insert、update和delete操作时，需要更新B+树索引。如果索引过多，会消耗很多额外的性能。

那么，问题来了，如果表中的索引太多，超过了5个该怎么办？

这个问题要辩证的看，如果你的系统并发量不高，表中的数据量也不多，其实超过5个也可以，只要不要超过太多就行。

但对于一些高并发的系统，请务必遵守单表索引数量不要超过5的限制。

那么，高并发系统如何优化索引数量？

能够建联合索引，就别建单个索引，可以删除无用的单个索引。

将部分查询功能迁移到其他类型的数据库中，比如：Elastic Seach、HBase等，在业务表中只需要建几个关键索引即可。

13 选择合理的字段类型

char 表示固定字符串类型，该类型的字段存储空间的固定的，会浪费存储空间。

```
alter table order
add column code char(20) NOT NULL;
```

varchar 表示变长字符串类型，该类型的字段存储空间会根据实际数据的长度调整，不会浪费存储空间。

```
alter table order
add column code varchar(20) NOT NULL;
```

如果是长度固定的字段，比如用户手机号，一般都是11位的，可以定义成char类型，长度是11字节。

但如果是企业名称字段，假如定义成char类型，就有问题了。

如果长度定义得太长，比如定义成了200字节，而实际企业长度只有50字节，则会浪费150字节的存储空间。

如果长度定义得太短，比如定义成了50字节，但实际企业名称有100字节，就会存储不下，而抛出异常。

所以建议将企业名称改成varchar类型，变长字段存储空间小，可以节省存储空间，而且对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

我们在选择字段类型时，应该遵循这样的原则：

1. 能用数字类型，就不用字符串，因为字符的处理往往比数字要慢。
2. 尽可能使用小的类型，比如：用bit存布尔值，用tinyint存枚举值等。

3. 长度固定的字符串字段，用char类型。
4. 长度可变的字符串字段，用varchar类型。
5. 金额字段用decimal，避免精度丢失问题。

还有很多原则，这里就不一一列举了。

14 提升group by的效率

我们有很多业务场景需要使用 `group by` 关键字，它主要的功能是去重和分组。

通常它会跟 `having` 一起配合使用，表示分组后再根据一定的条件过滤数据。

反例：

```
select user_id,user_name from order
group by user_id
having user_id <= 200;
```

这种写法性能不好，它先把所有的订单根据用户id分组之后，再去过滤用户id大于等于200的用户。

分组是一个相对耗时的操作，为什么我们不先缩小数据的范围之后，再分组呢？

正例：

```
select user_id,user_name from order
where user_id <= 200
group by user_id
```

使用where条件在分组前，就把多余的数据过滤掉了，这样分组时效率就会更高一些。

> 其实这是一种思路，不仅限于group by的优化。我们的sql语句在做一些耗时的操作之前，应尽可能缩小数据范围，这样能提升sql整体的性能。

15 索引优化

sql优化当中，有一个非常重要的内容就是： 索引优化 。

很多时候sql语句，走了索引，和没有走索引，执行效率差别很大。所以索引优化被作为sql优化的首选。

索引优化的第一步是：检查sql语句有没有走索引。

那么，如何查看sql走了索引没？

可以使用 `explain` 命令，查看mysql的执行计划。

例如：

```
explain select * from `order` where code='002';
```

结果：

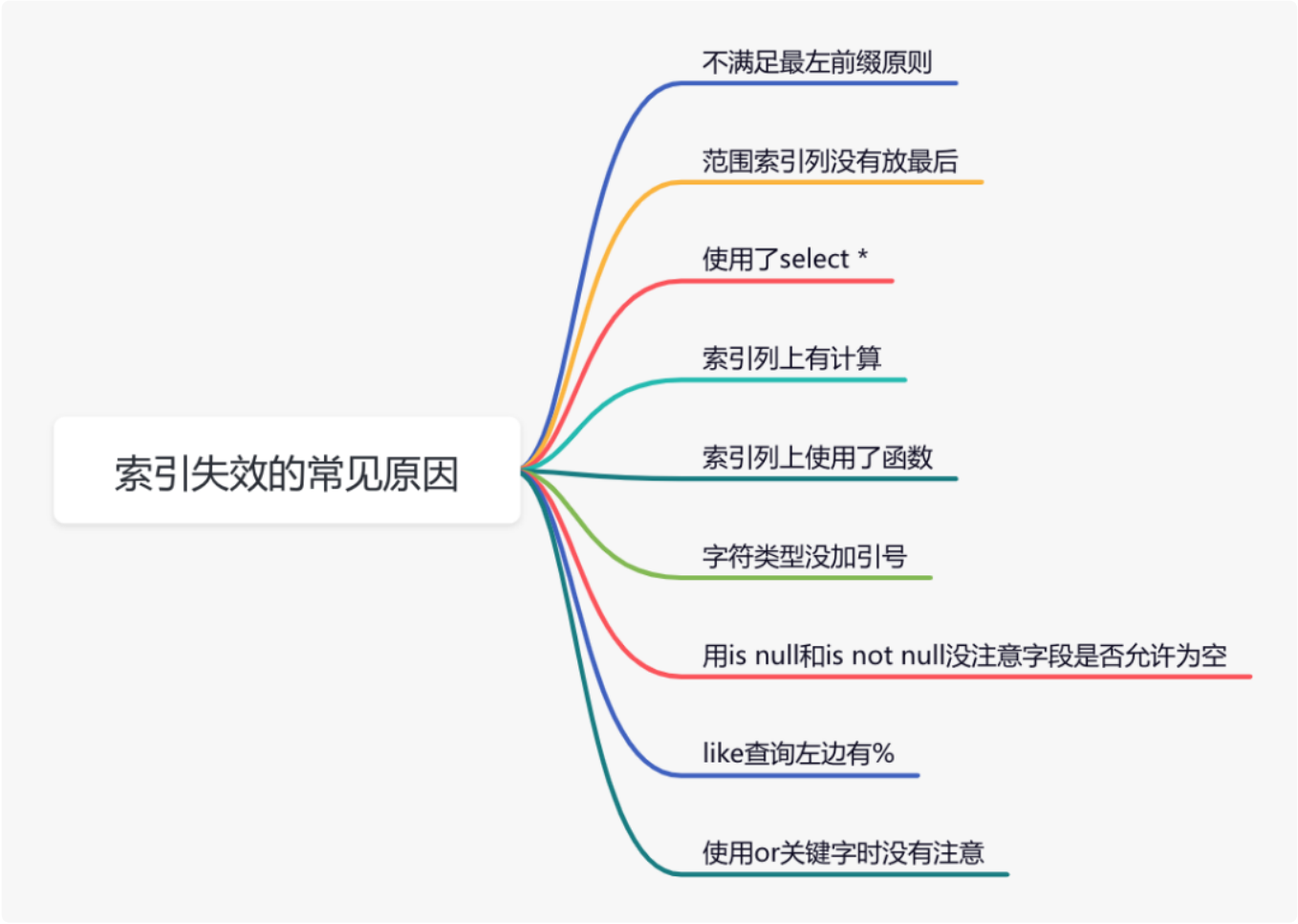
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(NULL)	const	un_code,un_code_name	un_code	82	const	1	100.00	(NULL)

通过这几列可以判断索引使用情况，执行计划包含列的含义如下图所示：


执行计划显示列
id (select唯一标识)
select_type (select类型)
table (表名称)
partitions (匹配的分区)
type (连接类型)
possible_keys (可能的索引选择)
key (实际用到的索引)
key_len (实际索引长度)
ref (与索引比较的列)
rows (预计要检查的行数)
filtered (按表条件过滤的行百分比)

说实话，sql语句没有走索引，排除没有建索引之外，最大的可能性是索引失效了。

下面说说索引失效的常见原因：



如果不是上面的这些原因，则需要再进一步排查一下其他原因。

此外，你有没有遇到过这样一种情况：明明是同一条sql，只有入参不同而已。有的时候走的索引a，有的时候却走的索引b？

没错，有时候mysql会选错索引。

必要时可以使用 `force index` 来强制查询sql走某个索引。

至于为什么mysql会选错索引，后面有专门的文章介绍的，这里先留点悬念。

最后说一句(求关注，别白嫖我)

如果这篇文章对您有所帮助，或者有所启发的话，帮忙扫描下发二维码关注一下，您的支持是我坚持写作最大的动力。

求一键三连：点赞、转发、在看。

程序员田螺

专注分享后端面试题，包括计算机网络、MySql数据库、Redis缓存、操作系统、Java后端、大厂面试真题等领域。

11篇原创内容

公众号

喜欢此内容的人还喜欢

一本助你起飞的Java编程书（文末送书）

捡田螺的小男孩