**1.避免使用select \***

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。

但我们写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。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。

**2.小表驱动大表**

小表驱动大表，即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)

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。

**3.避免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。

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。

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。

**4.高效的分页**

* • 有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;

* • 但是当数据多的时候，例如

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。

**解决方案**：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。

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

**5.连接代替子查询**

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

**6.控制join表的数量**

* • 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。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。
* • 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。
* • 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。

**7.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。

**8.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。

**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。

**9.用union/union all 代替or和<>**

* • or可能会造成索引失效，效率降低。
* • 根据情况用联合查询代替，利用索引提升效率

**10.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。
* • 索引优化的第一步是：检查sql语句有没有走索引。可以使用explain命令，查看mysql的执行计划。进而查看索引是否生效
* • sql语句没有走索引，排除没有建索引之外，最大的可能性就是**索引失效**。

