# SQL优化（系列）- 查询条件优化

**一、查询条件优化**

    SQL查询条件优化是提高查询性能的关键之一。以下是一些常见的 SQL 查询条件优化建议：

**1.1、使用合适的索引：**

    在查询中使用合适的索引可以显著提高查询性能。通过分析查询的 WHERE 条件和 JOIN 条件来确定哪些列适合建立索引，并使用 CREATE INDEX 语句在这些列上创建索引。

**1.2、避免使用通配符查询：**

    尽量避免使用通配符 % 和 \_ 进行模糊查询，因为这样的查询通常无法利用索引，会导致全表扫描而降低查询性能。

**1.3、使用 EXISTS 或 IN 替代 NOT EXISTS 或 NOT IN：**

    在有子查询的情况下，使用 EXISTS 或 IN 操作符比使用 NOT EXISTS 或 NOT IN 操作符性能更好。

**1.4、合理使用 LIMIT：**

    在不需要全部结果的情况下，可以使用 LIMIT 限制返回结果的数量，避免不必要的工作量和网络传输开销。

**1.5、避免在过滤字段上进行计算：**

    尽量避免在过滤字段上进行运算或函数操作，这会导致无法利用索引，降低查询性能。

**1.6、拆分复杂查询：**

    将复杂的查询拆分为多个简单的查询，可以减少查询的复杂度，提高性能和可读性。

**1.7、避免在 JOIN 条件上使用函数：**

    在 JOIN 条件中不要使用函数，可以影响查询的性能。如果需要使用函数，尽量将函数应用在需要结果的列上而不是在连接条件上。

**1.8、分析查询计划：**

    通过使用 EXPLAIN SELECT 命令来查看查询计划，可以帮助理解查询执行方式，发现潜在的性能瓶颈，并进行必要的优化。

以上是一些常用的 SQL 查询条件优化建议，通过合理优化查询条件，可以显著提升 Impala 查询的性能和效率。

**二、实践过程**

**1、分步执行效率高。**

    对于复杂的查询，可以使用中间临时表 暂存数据；

**2、优化group by语句**

    默认情况下，MySQL 会对GROUP BY分组的所有值进行排序，如 “GROUP BY col1，col2，....;” 查询的方法如同在查询中指定 “ORDER BY col1，col2，...;” 如果显式包括一个包含相同的列的 ORDER BY子句，MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。

因此，如果查询包括 GROUP BY 但你并不想对分组的值进行排序，你可以指定 ORDER BY NULL禁止排序。例如

SELECT col1, col2, COUNT(\*) FROM table GROUP BY col1, col2 ORDER BY NULL ;

**3、优化join语句**

    MySQL中可以通过子查询来使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 SQL 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接(JOIN)..替代。

例子：假设要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

SELECT col1 FROM customerinfo WHERE CustomerID NOT in (SELECT CustomerID FROM salesinfo )

    如果使用连接(JOIN).. 来完成这个查询工作，速度将会有所提升。尤其是当 salesinfo表中对 CustomerID 建有索引的话，性能将会更好，查询如下：

SELECT col1 FROM customerinfo LEFT JOIN salesinfo ON customerinfo.CustomerID=salesinfo.CustomerID WHERE salesinfo.CustomerID IS NULL

    连接(JOIN).. 之所以更有效率一些，是因为 MySQL 不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

**4、优化union查询**

    MySQL通过创建并填充临时表的方式来执行union查询。除非确实要消除重复的行，否则建议使用union all。原因在于如果没有all这个关键词，MySQL会给临时表加上distinct选项，这会导致对整个临时表的数据做唯一性校验，这样做的消耗相当高。

高效：

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL1 = 10   
UNION ALL   
SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';

低效：

SELECT COL1, COL2, COL3 FROM TABLE WHERE COL1 = 10   
UNION   
SELECT COL1, COL2, COL3 FROM TABLE WHERE COL3= 'TEST';

**5、拆分复杂SQL为多个小SQL，避免大事务**

* 简单的SQL容易使用到MySQL的QUERY CACHE；
* 减少锁表时间特别是使用MyISAM存储引擎的表；
* 可以使用多核CPU。

**6、使用truncate代替delete**

    当删除全表中记录时，使用delete语句的操作会被记录到undo块中，删除记录也记录binlog，当确认需要删除全表时，会产生很大量的binlog并占用大量的undo数据块，此时既没有很好的效率也占用了大量的资源。

    使用truncate替代，不会记录可恢复的信息，数据不能被恢复。也因此使用truncate操作有其极少的资源占用与极快的时间。另外，使用truncate可以回收表的水位，使自增字段值归零。

**7、使用合理的分页方式以提高分页效率**

    使用合理的分页方式以提高分页效率 针对展现等分页需求，合适的分页方式能够提高分页的效率。

案例1：

select \* from t where thread\_id = 10000 and deleted = 0 order by gmt\_create asc limit 0, 15;

    上述例子通过一次性根据过滤条件取出所有字段进行排序返回。数据访问开销=索引IO+索引全部记录结果对应的表数据IO。因此，该种写法越翻到后面执行效率越差，时间越长，尤其表数据量很大的时候。

    适用场景：当中间结果集很小（10000行以下）或者查询条件复杂（指涉及多个不同查询字段或者多表连接）时适用。

案例2：

select t.\* from (select id from t where thread\_id = 10000 and deleted = 0order by gmt\_create asc limit 0, 15) a, t where a.id = t.id;

    上述例子必须满足t表主键是id列，且有覆盖索引secondary key:(thread\_id, deleted, gmt\_create)。通过先根据过滤条件利用覆盖索引取出主键id进行排序，再进行join操作取出其他字段。数据访问开销=索引IO+索引分页后结果（例子中是15行）对应的表数据IO。因此，该写法每次翻页消耗的资源和时间都基本相同，就像翻第一页一样。

    适用场景：当查询和排序字段（即where子句和order by子句涉及的字段）有对应覆盖索引时，且中间结果集很大的情况时适用。