**sql语句的执行顺序:**

1. FROM <left\_table>
2. ON <join\_condition>
3. <join\_type> JOIN <right\_table>
4. WHERE <where\_condition>
5. GROUP BY <group\_by\_list>
6. HAVING <having\_condition>
7. SELECT
8. DISTINCT <select\_list>
9. ORDER BY <order\_by\_condition>
10. LIMIT <limit\_number>

**1. LIMIT 语句**

分页查询是最常用的场景之一，但也通常也是最容易出问题的地方。比如对于下面简单的语句，一般 DBA 想到的办法是在 type, name, create\_time 字段上加组合索引。这样条件排序都能有效的利用到索引，性能迅速提升。

SELECT \* FROM   operation WHERE  type = 'SQLStats'

AND name = 'SlowLog' ORDER  BY create\_time LIMIT  1000, 10;

好吧，可能90%以上的 DBA 解决该问题就到此为止。但当 LIMIT 子句变成 “LIMIT 1000000,10” 时，程序员仍然会抱怨：我只取10条记录为什么还是慢？

要知道数据库也并不知道第1000000条记录从什么地方开始，即使有索引也需要从头计算一次。出现这种性能问题，多数情形下是程序员偷懒了。

在前端数据浏览翻页，或者大数据分批导出等场景下，是可以将上一页的最大值当成参数作为查询条件的。SQL 重新设计如下：

SELECT \* FROM operation WHERE type = 'SQLStats'

AND name = 'SlowLog' AND create\_time > '2017-03-16 14:00:00'

ORDER BY create\_time limit 10;

在新设计下查询时间基本固定，不会随着数据量的增长而发生变化。

**2. 隐式转换**

SQL语句中查询变量和字段定义类型不匹配是另一个常见的错误。比如下面的语句：

mysql> explain extended SELECT \* FROM  my\_balance b

   > WHERE  b.bpn = 14000000123

   >       AND b.isverified IS NULL ;

mysql> show warnings;

| Warning | 1739 | Cannot use ref access on index 'bpn' due to type or collation conversion on field 'bpn'

其中字段 bpn 的定义为 varchar(20)，MySQL 的策略是将字符串转换为数字之后再比较。函数作用于表字段，索引失效。

上述情况可能是应用程序框架自动填入的参数，而不是程序员的原意。现在应用框架很多很繁杂，使用方便的同时也小心它可能给自己挖坑。

**3. 关联更新、删除**

虽然 MySQL5.6 引入了物化特性，但需要特别注意它目前仅仅针对查询语句的优化。对于更新或删除需要手工重写成 JOIN。

比如下面 UPDATE 语句，MySQL 实际执行的是循环/嵌套子查询（DEPENDENT SUBQUERY)，其执行时间可想而知。

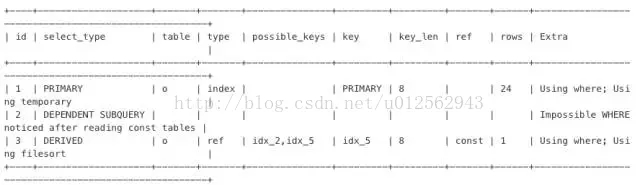
UPDATE operation o SET status = 'applying' WHERE  o.id

IN (SELECT id FROM (SELECT o.id,o.status FROM   operation o

WHERE  o.group = 123 AND o.status NOT IN ( 'done' )

ORDER  BY o.parent, o.id LIMIT  1) t);

 执行计划：



重写为 JOIN 之后，子查询的选择模式从 DEPENDENT SUBQUERY 变成 DERIVED，执行速度大大加快，从7秒降低到2毫秒。

UPDATE operation o JOIN  (SELECT o.id, o.status FROM   operation o WHERE  o.group = 123

AND o.status NOT IN ( 'done' ) ORDER  BY o.parent,o.id LIMIT  1) t

ON o.id = t.id SET    status = 'applying'

执行计划简化为：



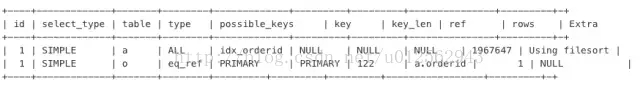
**4. 混合排序**

MySQL 不能利用索引进行混合排序。但在某些场景，还是有机会使用特殊方法提升性能的。

SELECT \* FROM my\_order o INNER JOIN my\_appraise a ON a.orderid = o.id

ORDER  BY a.is\_reply ASC, a.appraise\_time DESC LIMIT  0, 20

执行计划显示为全表扫描：



由于 is\_reply 只有0和1两种状态，我们按照下面的方法重写后，执行时间从1.58秒降低到2毫秒。

SELECT \* FROM (

(SELECT \* FROM my\_order o INNER JOIN my\_appraise a  ON a.orderid = o.id

AND is\_reply = 0 ORDER  BY appraise\_time DESC LIMIT  0, 20) 

UNION ALL

(SELECT \* FROM my\_order o INNER JOIN my\_appraise a ON a.orderid = o.id

AND is\_reply = 1 ORDER  BY appraise\_time DESC LIMIT  0, 20)) t

ORDER  BY  is\_reply ASC, appraisetime DESC LIMIT  20;

**5. EXISTS语句**

MySQL 对待 EXISTS 子句时，仍然采用嵌套子查询的执行方式。如下面的 SQL 语句：

SELECT \*

FROM   my\_neighbor n

     LEFT JOIN my\_neighbor\_apply sra

            ON n.id = sra.neighbor\_id

               AND sra.user\_id = 'xxx'

WHERE  n.topic\_status < 4

     AND EXISTS(SELECT 1

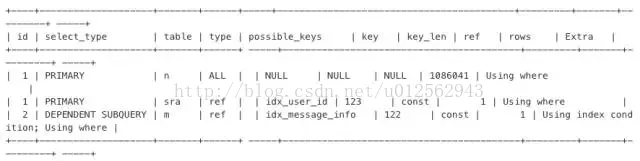
                FROM   message\_info m

                WHERE  n.id = m.neighbor\_id

                       AND m.inuser = 'xxx')

     AND n.topic\_type <> 5

执行计划为：



去掉 exists 更改为 join，能够避免嵌套子查询，将执行时间从1.93秒降低为1毫秒。

SELECT \*

FROM   my\_neighbor n

     INNER JOIN message\_info m

             ON n.id = m.neighbor\_id

                AND m.inuser = 'xxx'

     LEFT JOIN my\_neighbor\_apply sra

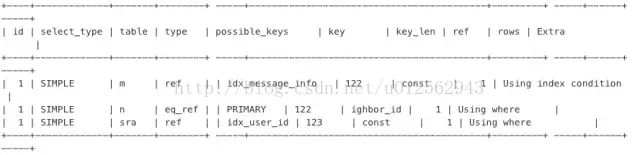
            ON n.id = sra.neighbor\_id

               AND sra.user\_id = 'xxx'

WHERE  n.topic\_status < 4

     AND n.topic\_type <> 5

新的执行计划：



**6. 条件下推**

外部查询条件不能够下推到复杂的视图或子查询的情况有：

1、聚合子查询；

2、含有 LIMIT 的子查询；

3、UNION 或 UNION ALL 子查询；

4、输出字段中的子查询；

如下面的语句，从执行计划可以看出其条件作用于聚合子查询之后：

SELECT \*

FROM   (SELECT target,

             Count(\*)

      FROM   operation

      GROUP  BY target) t

WHERE  target = 'rm-xxxx'



确定从语义上查询条件可以直接下推后，重写如下：

SELECT target,

     Count(\*)

FROM   operation

WHERE  target = 'rm-xxxx'

GROUP  BY target

执行计划变为：

IMG_256

**7. 提前缩小范围**

先上初始 SQL 语句：

SELECT \*

FROM   my\_order o

     LEFT JOIN my\_userinfo u

            ON o.uid = u.uid

     LEFT JOIN my\_productinfo p

            ON o.pid = p.pid

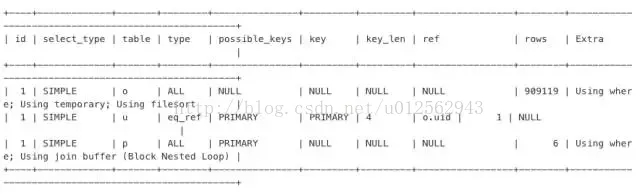
WHERE  ( o.display = 0 )

     AND ( o.ostaus = 1 )

ORDER  BY o.selltime DESC

LIMIT  0, 15

该SQL语句原意是：先做一系列的左连接，然后排序取前15条记录。从执行计划也可以看出，最后一步估算排序记录数为90万，时间消耗为12秒。



由于最后 WHERE 条件以及排序均针对最左主表，因此可以先对 my\_order 排序提前缩小数据量再做左连接。SQL 重写后如下，执行时间缩小为1毫秒左右。

SELECT \*

FROM (

SELECT \*

FROM   my\_order o

WHERE  ( o.display = 0 )

     AND ( o.ostaus = 1 )

ORDER  BY o.selltime DESC

LIMIT  0, 15

) o

   LEFT JOIN my\_userinfo u

            ON o.uid = u.uid

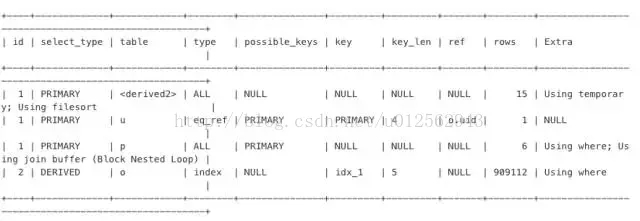
   LEFT JOIN my\_productinfo p

            ON o.pid = p.pid

ORDER BY  o.selltime DESC

limit 0, 15

再检查执行计划：子查询物化后（select\_type=DERIVED)参与 JOIN。虽然估算行扫描仍然为90万，但是利用了索引以及 LIMIT 子句后，实际执行时间变得很小。



**8. 中间结果集下推**

再来看下面这个已经初步优化过的例子(左连接中的主表优先作用查询条件)：

SELECT    a.\*,

        c.allocated

FROM      (

            SELECT   resourceid

            FROM     my\_distribute d

                 WHERE    isdelete = 0

                 AND      cusmanagercode = '1234567'

                 ORDER BY salecode limit 20) a

LEFT JOIN

        (

            SELECT   resourcesid， sum(ifnull(allocation, 0) \* 12345) allocated

            FROM     my\_resources

                 GROUP BY resourcesid) c

ON        a.resourceid = c.resourcesid

那么该语句还存在其它问题吗？不难看出子查询 c 是全表聚合查询，在表数量特别大的情况下会导致整个语句的性能下降。

其实对于子查询 c，左连接最后结果集只关心能和主表 resourceid 能匹配的数据。因此我们可以重写语句如下，执行时间从原来的2秒下降到2毫秒。

SELECT    a.\*,

        c.allocated

FROM      (

                 SELECT   resourceid

                 FROM     my\_distribute d

                 WHERE    isdelete = 0

                 AND      cusmanagercode = '1234567'

                 ORDER BY salecode limit 20) a

LEFT JOIN

        (

                 SELECT   resourcesid， sum(ifnull(allocation, 0) \* 12345) allocated

                 FROM     my\_resources r,

                          (

                                   SELECT   resourceid

                                   FROM     my\_distribute d

                                   WHERE    isdelete = 0

                                   AND      cusmanagercode = '1234567'

                                   ORDER BY salecode limit 20) a

                 WHERE    r.resourcesid = a.resourcesid

                 GROUP BY resourcesid) c

ON        a.resourceid = c.resourcesid

但是子查询 a 在我们的SQL语句中出现了多次。这种写法不仅存在额外的开销，还使得整个语句显的繁杂。使用 WITH 语句再次重写：

WITH a AS

(

       SELECT   resourceid

       FROM     my\_distribute d

       WHERE    isdelete = 0

       AND      cusmanagercode = '1234567'

       ORDER BY salecode limit 20)

SELECT    a.\*,

        c.allocated

FROM      a

LEFT JOIN

        (

                 SELECT   resourcesid， sum(ifnull(allocation, 0) \* 12345) allocated

                 FROM     my\_resources r,

                          a

                 WHERE    r.resourcesid = a.resourcesid

                 GROUP BY resourcesid) c

ON        a.resourceid = c.resourcesid

**总结**

数据库编译器产生执行计划，决定着SQL的实际执行方式。但是编译器只是尽力服务，所有数据库的编译器都不是尽善尽美的。

上述提到的多数场景，在其它数据库中也存在性能问题。了解数据库编译器的特性，才能避规其短处，写出高性能的SQL语句。

程序员在设计数据模型以及编写SQL语句时，要把算法的思想或意识带进来。

编写复杂SQL语句要养成使用 WITH 语句的习惯。简洁且思路清晰的SQL语句也能减小数据库的负担 。