查询性能优化

[TOC]

一、优化规则

1.1 索引优化

1.2 查询优化（SQL）

1.3 库表结构优化

二、SQL语句优化实战

2.1 使用explain

案例1

1. 原始语句

```
EXPLAIN SELECT
        IFNULL(a.c1, 0) AS cpctotalnum,
        IFNULL(b.c2, 0) AS cpcgrpnum,
        IFNULL(c.c3, 0) AS keylenbig,
        (IFNULL(a.c1, 0)) - IFNULL(c.c3, 0) AS keylensmall,
        IFNULL(cp.plancnt, 0) AS cpcgrpnum,
        IFNULL(d.c4, 0) AS unioncpcplannum,
        a.accountid
FROM
        (
                SELECT
                        count(*) c1,
                        accountid
                FROM
                        cpc
                WHERE
                        check_status <> - 1
                AND ispause = 0
                AND accountid = { accountid }
                GROUP BY
                        accountid
        ) a
LEFT JOIN (
        SELECT
                count(*) c2,
                accountid
        FROM
                cpcgrp
        WHERE
                accountid = { accountid }
```

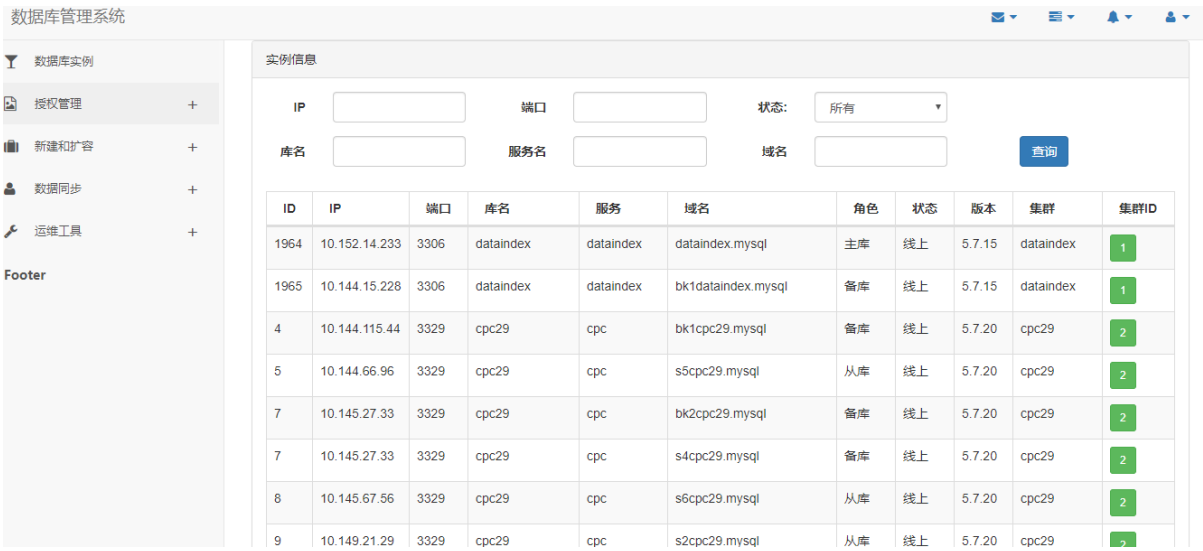
```
        GROUP BY
                accountid
) b ON a.accountid = b.accountid
LEFT JOIN (
        SELECT
                count(*) plancnt,
                accountid
        FROM
                cpcplan
        WHERE
                accountid = { accountid }
        GROUP BY
                accountid
) cp ON a.accountid = cp.accountid
LEFT JOIN (
        SELECT
                count(*) c3,
                accountid
        FROM
                cpc
        WHERE
                check_status <> - 1
        AND ispause = 0
        AND length(`key`) > 10
        AND accountid = { accountid }
        GROUP BY
                accountid
) c ON a.accountid = cp.accountid
LEFT JOIN (
        SELECT
                count(*) c4,
                accountid
        FROM
                cpcplan
        WHERE
                isunion_show = 1
        AND accountid = { accountid }
        GROUP BY
                accountid
) d ON a.accountid = cp.accountid
```

2. 测试

一些说明：

- 使用旭日cpc库，32*32，online-s1线上备库，进行select语句分析。
- 使用了cpc-29-01分库分表（这里顺便学习了如何通过账户ID计算器所在的库表）。
- 实践了上节课老师讲解的“如何使用数据库管理系统”，查找相关数据库的地址：
    - 网址：http://dbmall2.sogou/instance/

- 界面：



- 实际使用也是找对应的库表进行相关查询。

explain结果分析：

```
60  SELECT
61    count(*) c4,
62    accountid
63  FROM
64    cpcplan_2901
65  WHERE
66    isunion_show = 1
67  AND accountid = 18614300
68  GROUP BY
69    accountid
70  ) d ON a.accountid = cp.accountid
```

信息 结果1 概况 状态

id	select_type	table	partitior	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>	(Null)	system	(Null)	(Null)	(Null)	(Null)	1	100	(Null)
1	PRIMARY	<derived3>	(Null)	system	(Null)	(Null)	(Null)	(Null)	1	100	(Null)
1	PRIMARY	<derived4>	(Null)	system	(Null)	(Null)	(Null)	(Null)	1	100	(Null)
1	PRIMARY	<derived5>	(Null)	system	(Null)	(Null)	(Null)	(Null)	1	100	(Null)
1	PRIMARY	<derived6>	(Null)	system	(Null)	(Null)	(Null)	(Null)	1	100	(Null)
6	DERIVED	cpcplan_2901	(Null)	ref	uq_aid_name_type	uq_aid_name_type	4	const	7	10	Using where
5	DERIVED	cpc_2901	(Null)	ref	ix_cpc_aid_pid_grpid,ix_cpc_aid_key	ix_cpc_aid_pid_grpid	4	const	100132	9	Using where
4	DERIVED	cpcplanlevel_2901	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	8960	10	Using where
3	DERIVED	cpcgrp_2901	(Null)	ref	ix_cpcgrp_aid	ix_cpcgrp_aid	4	const	33	100	Using index
2	DERIVED	cpc_2901	(Null)	ref	ix_cpc_aid_pid_grpid,ix_cpc_aid_key	ix_cpc_aid_pid_grpid	4	const	100132	9	Using where

EXPLAIN SELECT IFNULL(a.c1, 0) AS cpctotalnum, IFNULL(b.c2, 0) AS cpcgrpnum, IFNULL(c.c3, 0) 只读 查询时间: 0.620s 第 1 条记录 (共 10 条)

- **select_type**:简单语句会是SIMPLE（不包括子查询和UNION），这里我们分析复杂语句，所以一般见不到这个，复杂语句中最外层的部分标记为**PRIMARY**，其他部分主要有三类：
  - **SUBQUERY** 不在FROM字句中的SELECT；
  - **DERIVED** 在FROM字句中的SELECT；
  - **UNION** 在UNION中的第二个和随后的SELECT都被标记为UNION，那么第一个呢？看情况，可能是前面两者中一个（SUBQUERY或者DERIVED），取决于UNION被不被FROM字句所包含。

> 本例只有PRIMARY和DERIVED。后者在FROM字句中出现。

- **table**：显示的是对应访问哪个表 。后面几个具体的表名还比较清晰，但是前面的<derivedN>是什么东西？ 其实，当在FROM子句中有子查询时，table列就会是这个形式，其中N是子查询的id。
- **type**:访问类型——MySQL决定如何查找表中的行，访问方法从最差到最优分别为：
  - ALL 全表扫描
  - index 跟全表扫描一样，不过是按索引次序进行而不是行，主要优点是避免了排序。

    - range 范围扫描，有限制的索引扫描，常见于有BETWEEN，WHERE字句带有>的查询，或者IN(),OR列表等。
    - ref 索引访问，是查找和扫描的混合体，索引要跟某个参考值进行比较。
    - system、const 表示能对查询的某部分进行优化并可转换成一个常量时。
- **possible_keys**：可能会用到的索引。
- **key**：实际用到的索引。

> 重点看没用到索引的子句，本例子中，**cpcplanlevel_2901**表示没有建立索引的，这里是一个**潜在的优化点**，可以看看有没有必要建立一个主键索引？

- **key_len**:索引使用的字节数。
- **ref**： 显示之前的表在key列记录的索引中查找值所用到的列或常量。
- **rows**： MySQL为了找到符合查询的每一个点上标准的那些行而必须读取的行的平均数。

> 可以看到本例中，有两个查询可能读到十万行，相对比较大，其他都比较小，重点看需要读取很多行的查询子句。

- **Extra**：一些不适合在其他列显示的额外信息：
    - Using index ： 将使用覆盖索引，以避免访问表。
    - Using where : MySQL服务器将在存储引擎检索行后再进行过滤。
    - Using temporary ： 在对查询结果排序时使用了一个临时表。

3. 总结： 本例查询行数最多是10万，时间是0.6s，而且大部分时间是在send data步骤，实际执行时间只有

信息 结果1 概况 状态

状态	期间	百分比
starting	0.0001610	0.065%
checking permissions	0.0000060	0.002%
Opening tables	0.0000170	0.007%
init	0.0000780	0.031%
System lock	0.0000070	0.003%
optimizing	0.0000570	0.023%
statistics	0.0003000	0.120%
preparing	0.0001070	0.043%
executing	0.0000050	0.002%
Sending data	0.2482090	99.588%
explaining	0.0001310	0.053%
end	0.0000030	0.001%
query end	0.0000110	0.004%
closing tables	0.0000160	0.006%
removing tmp table	0.0000680	0.027%
freeing items	0.0000410	0.016%
cleaning up	0.0000180	0.007%

微秒： 可优化的点：

    - **cpcplanlevel_2901**表示没有建立索引，可以考虑建立主键索引。
    - 子查询过多，可以拆分，但本例子查询所花费时间并不复杂，耗时也不多（我单独运行过每个子句），所以这个方案可以排除。

案例2

1. 原始语句

```
SELECT
        xmlstyleid,
        dummyaccountid,
        checkstatus,
        xmltype,
        xmlstyletype,
        device_type,
        content,
        LEVEL,
        levelids,
        bidwords
FROM
        xmlstyle
WHERE
        xmlstyletype = 6
```

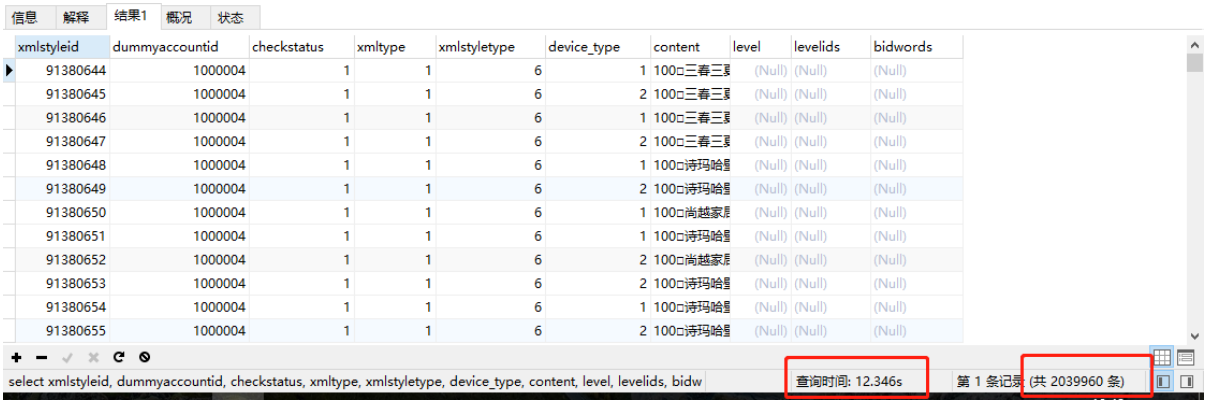
2. 测试 使用的是线上备库：

> 660 10.149.41.67 3365 xmlcpc xmlcpc bk1xmlcpc.mysql 备库 线上 5.5.27 xmlcpc

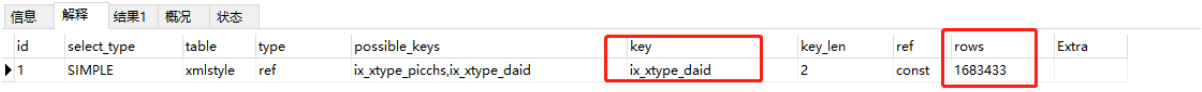
- 此条语句来自于biz_dba@sogou-inc.com的查询慢报警邮件：bizdev_xuri slow log top 20 (2019-08-20-2019-08-21) 。
- hostname:sxmlcpc
- db_name:xmlcpc
- count: 29
- avg_time(s): 13.8291
- sql语句:

```
  select xmlstyleid, dummyaccountid, checkstatus, xmltype,
xmlstyletype, device_type, content, level, levelids, bidwords from
xmlstyle where xmlstyletype = 6
```

- 我的查询结果：



- 我的explain分析



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	xmlstyle	ref	ix_xtype_picchs,ix_xtype_daid	ix_xtype_daid	2	const	1683433	

  - 索引用到了，不过这个索引不是在where子句的筛选条件中，具体有没有作用待进一步分析。
  - 查询数据超过百万，结果都有两百万条，耗时12s，还是偏多的。

3. 总结
   - 通过实际数据查看，xmlstyletype字段只有两种值，6或者7，所以这条语句约等于需要返回全部数据（一半）。
   - 而经过实际耗时分析，99.998%的时间都是在Sending data操作上，也就是说：耗时长跟sql语句的效率关系不大，主要在于数据传输上，这个数据传输优化是另外一个议题。

案例3

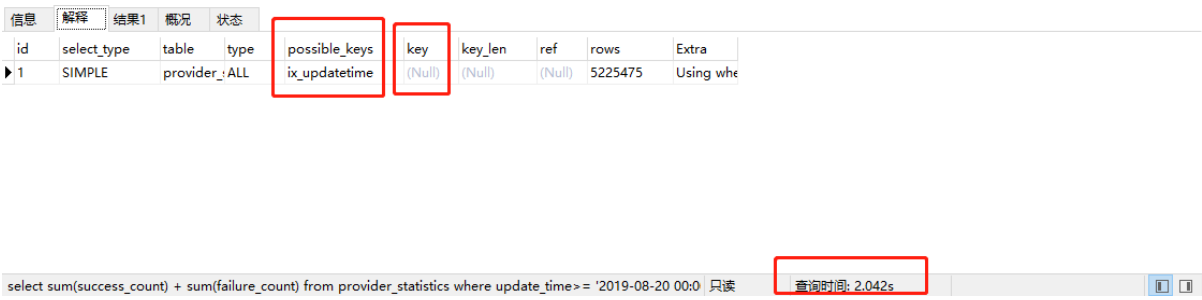
1. 原始语句
   - hostname: registry
   - db_name: registry
   - count: 25
   - avg_time(s): 3.19974
   - sql语句:

```
SELECT
        sum(success_count) + sum(failure_count)
FROM
        provider_statistics
WHERE
        update_time >= '2019-08-20 00:00:00'
AND     update_time < '2019-08-20 19:27:32'
```

2. 测试
   - 使用线上备库：

> 402 10.144.33.77 3306 registry registry bk1soa.mysql 备库 线上 5.5.27 registry

   - 测试结果：



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	provider_	ALL	ix_updatetime	(Null)	(Null)	(Null)	5225475	Using whe

select sum(success_count) + sum(failure_count) from provider_statistics where update_time>= '2019-08-20 00:0 只读 查询时间: 2.042s

   - 结果分析：
     - 预计用到索引：updatetime，实际没有用到，耗时2s
     - 如何让时间段的范围查询用到时间索引呢？这是优化方向。

3. 优化方向

- 是不是改成时间戳就能到时间索引？

> 查阅资料发现：将sql中的时间戳转化为日期能提高速度，可以让索引type从index转变成range，更优。我们已经是日期形式了，没必要再转回去。

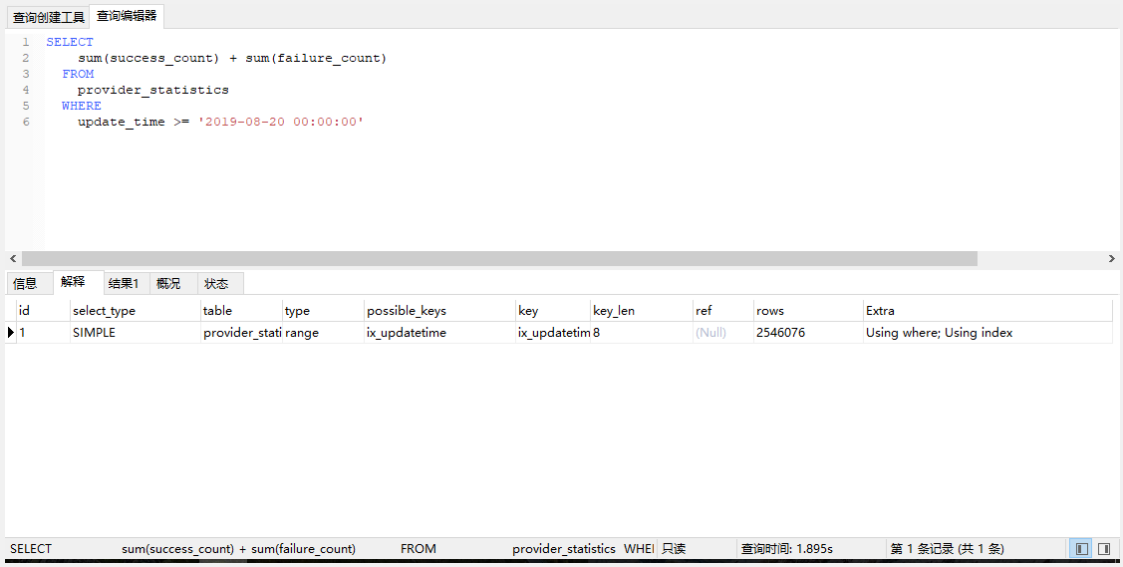
- 那会不会是因为在SELECT的字段中使用了sum函数？

> 尝试了去掉sum，只查询count，还是没用到索引

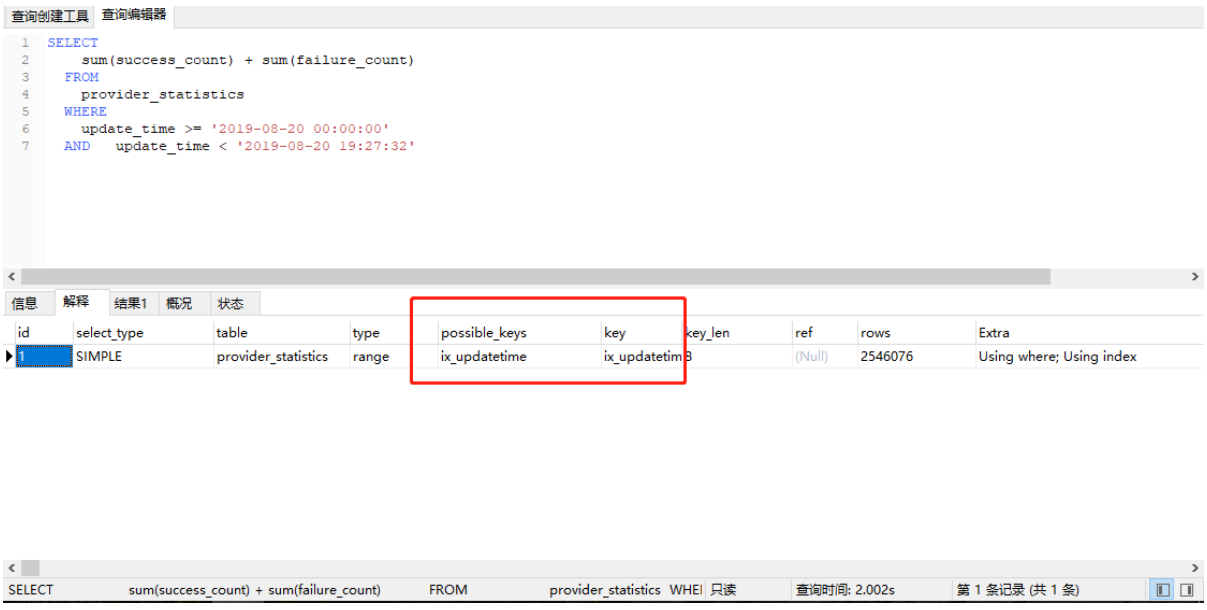
- 如果是查询id字段就能使用到这个索引？

- 这条sql的业务逻辑为统计出最近七天该表的数据量，可以去掉右边的小于等于 执行sql：

> 查询时间：从2.002s——>1.894s



- 突然发现，原来的语句是又能用到索引了，纳闷？？？



- 另外一种优化：新建一个**bigint**类型字段update_time_long存储update_time的毫秒值，并在update_time_long字段上建立索引,这个新增字段在开发库上进行。 其主要思想是：在**InnoDB**存储引擎下，比较**bigint**的效率高于**datetime**

4. 总结

   - 本例子中应该是能用到索引的，至于为什么最开始没用到，也许是工具显示错误，也许是其他原因，待进一步复现。
   - 由于是时间作为索引，根据业务实际需求，查询最近N天，如果起始时间就在N天内，那么后面的小于当前时间的条件可以去掉，能缩短一些时间。
   - 最重要的是能把type从index变为range，可以大大缩短时间'2019-08-20 00:00:00'，使用时间比使用时间戳1565456461（随便整的）更优。前者是range
   - 另外对于时间，转成bigint字段可以比datetime更快，因为比较bigint效率高于比较datetime。

使用**Explain**总结

1. 首先看type，有没有用到索引，ALL全表是最差的，要避免，最好是能优化到ref类型。
2. 然后看key用没用上，用上了哪个？没用上那可能会用上的有哪些？如何去修改成能用上。
3. 再看索引长度，短点会好些。
4. 影响行数多的查询优化的收益会更高。
5. 拆分复杂查询成小查询也是可以尝试的，现在的网络传输没那么差，可以看实际情况分析。