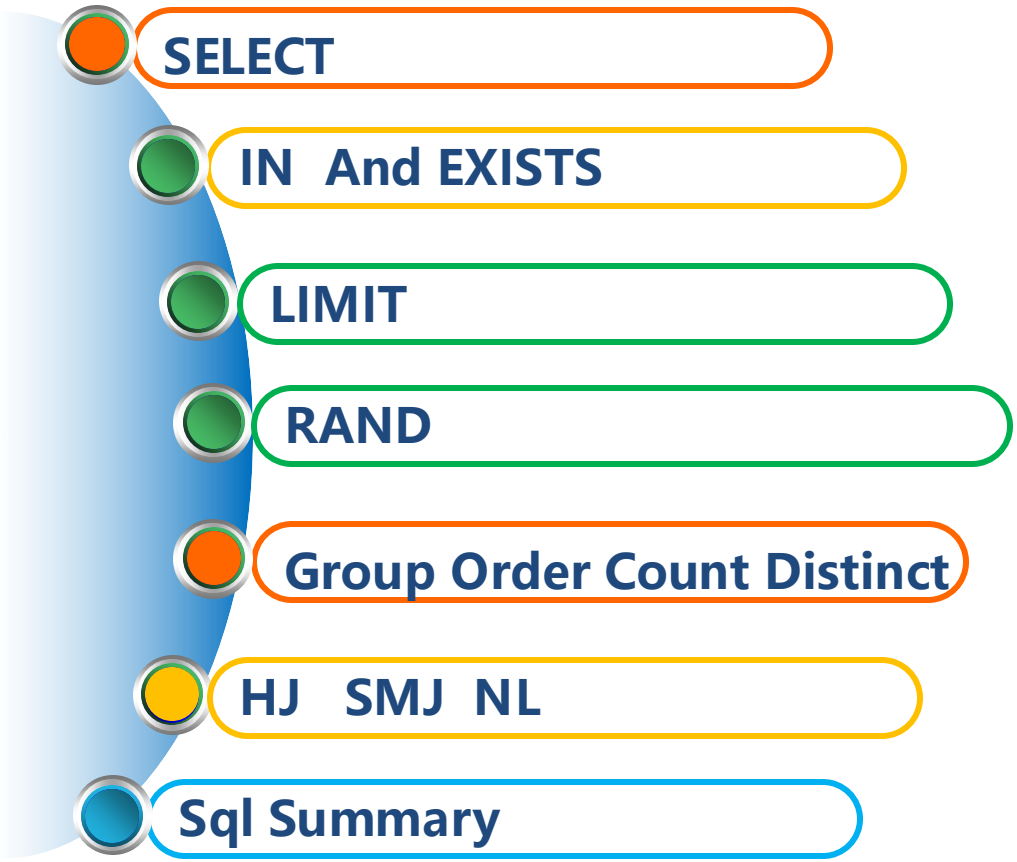
MySQL数据库优化之SQL篇

运维与安全部　数据管理服务

2012

MySQL SQL Tuning



Preface

数据库调优

对象	影响因素
数据库层次	表结构
	SQL 语句
	参数设置
硬件层次	CPU
	IO
	Memory
	NetWork

Preface

数据库层次

对象	影响因素
表	列数、数据类型
列	列格式是否合理
锁	锁策略是否合理
索引	表是否有合适索引结构
存储引擎	表的存储引擎是否合适
SQL	SQL 拼写
CACHE	缓存设置是否合理

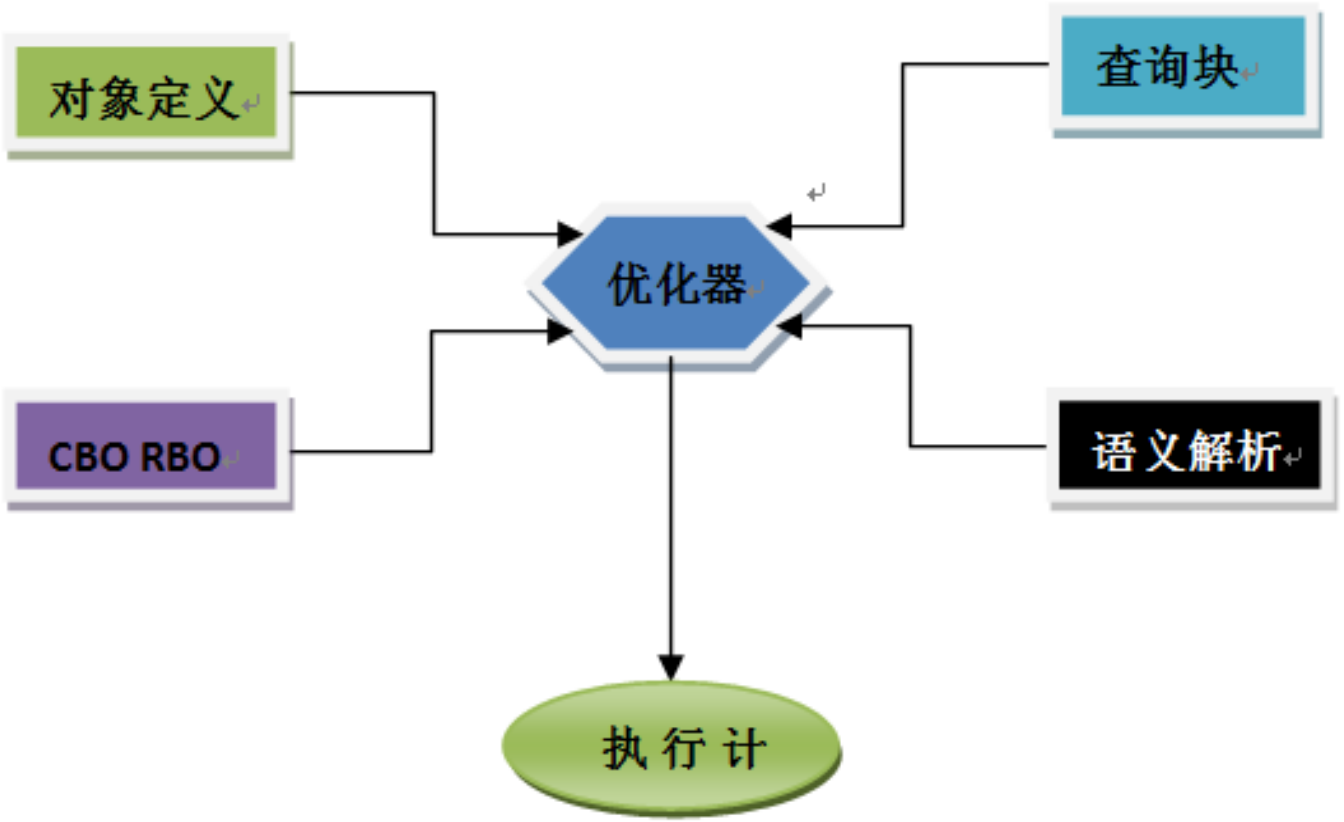
Preface

硬件层次

对象	影响因素
CPU	CPU 频率
Memory	内存大小
IO	IO 速度
NetWork	网卡速率

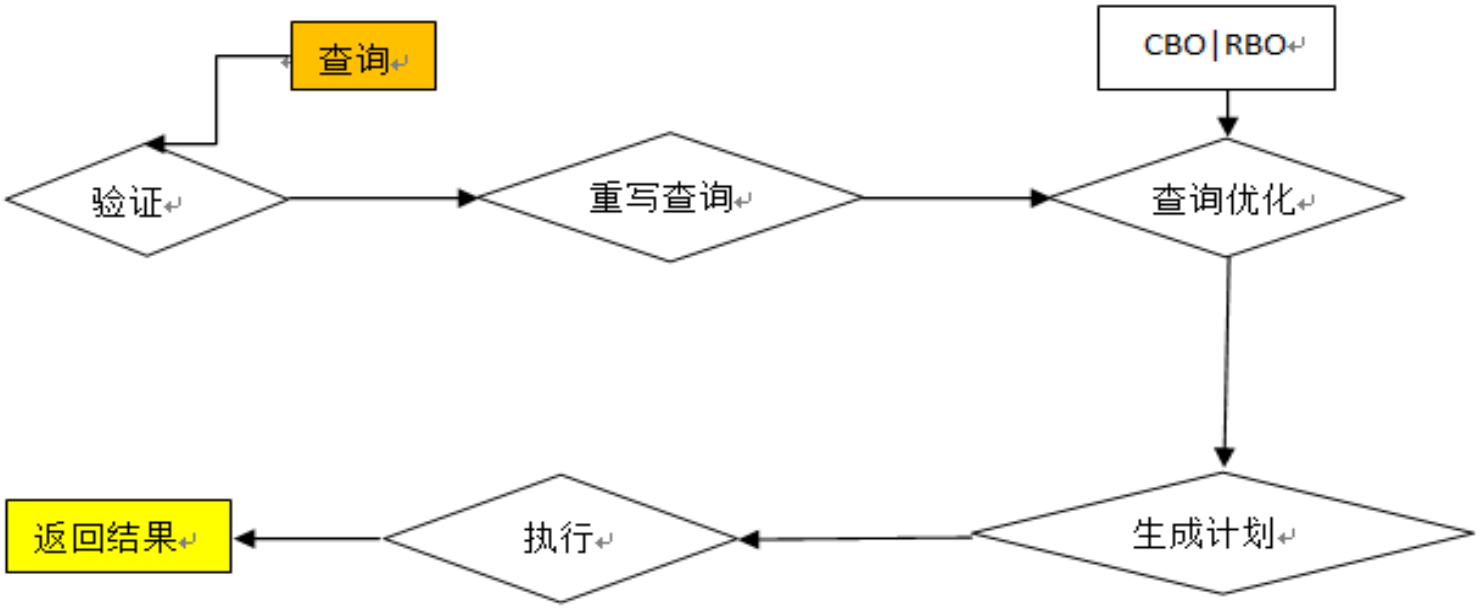
Preface

执行计划



Preface

CBO:基于对象/系统统计信息
RBO:基于规则优先级别.



操作阶段	描述
验证	权限检查、语法语义检查及简单转换操作
查询重写	OR 进行展开、视图、子查询合并
优化	决定访问路径、关联方式
生成计划	生成执行计划描述
执行	根据上一步生成计划，执行查询操作
返回结果	给客户端返回结果

Preface

SELECT

对象	影响因素
SELECT	针对出现 where 语句中的列创建合适索引
	避免全表扫描、尤其是针对大表
	适时更新表的统计信息
	尽量避免函数调用
	避免转换的查询方式
	针对 explain 调整执行计划
	配置合理的 cache 大小,用于缓存
	针对语句，进行最大优化，以提高并发
	考虑锁问题

SELECT

SELECT * ↵

- 选择表所有的列 ↵
- 对磁盘 IO 没有影响，因为 MySQL 读取数据是按页读取. ↵
- 从 CPU 角度考虑，可能利于减轻 CPU 负担.因为不进行分解操作.↵
- 影响网卡流量，因为一条记录要完整通过网络发送给查询端 ↵
- 对客户端有一定影响 ↵

但在以下情况应该禁止 SELECT * 语句 ↵

- 表中包含大字段 TEXT、BLOB↵

↵

↵

因为大字段独立行数据行外存储，对数据库端会造成额外 IO 开销.并且对 buffer 也造成一定压力. ↵

IN or EXISTS

关键字	描述
IN	可以是子查询或常量.但两者采用算法不同. 区分空值(NULL) 基于数值比较,需要指明特定条件
EXISTS	子查询子句 不区分空值(NULL) 基于 true 或 flase 判断,不需要指明确定条件

- 两不同，IN 区分空值,EXISTS 不分区空值.
- 针对子查询，采用算法是相同的算法
- IN 子查询在一定条件 MySQL 优化会重写改为 EXISTS 形式

IN or EXISTS

EXISTS

exists 不处理空值，就是 NULL 与 FLASE 是同义词.

exists 条件就是检查子查询返回 true 与 flase，然后来根据返 true 与 flase 来返回查询结果.

所以在子查询中，不需要指明具体的列值.当然也可指定

```
SELECT col1,col2,..,coln FROM  t1
WHERE exists  (SELECT * FROM T2 WHERE T1.col1=t2.col2 )
```

执行过程:

第一步: 选 t1 读取第一条记录的 t1.col 传递给子查询

第二步: t2 接受外层查询传来的 t1.col1 进行比较、匹配

第三步: 通过 t1.col1=t2.col2 进行判断.然后匹配成功就返回 true,否则就 flase

第四步: 外层查询接受 true 然后就打印出记录行，如果 flase 则丢弃。

第五步: 继续重复 1-4，直到全部处理完毕.

IN or EXISTS

IN

处理空值.null 可以被接受.in 是基于数据值比较，子查询要有数据列. MySQL 中处理 IN 与 exists 是一样的.从外层->内层 进行处理

```
SELECT col1,col2,..,coln  FROM  t1
WHERE t1.col1 IN (SELECT t2.col1 FROM T2)
```

第一步: 选 t1 读取第一条记录的 t1.col 传递给子查询

第二步: t2 接受外层查询传来的 t1.col1 进行比较、匹配

第三步: 通过 t1.col1=t2.col1 进行判断.然后匹配成功就返回 t1 记录.否则丢弃

第四步: 继续重复 1-3，直到全部处理完毕.

IN or EXISTS

IN 转换成 EXISTS,并且 push down。需要满足以下两个条件:

- 外部查询与内部查询都不能为 NULL
- NULL 与 FALSE 作相同对待，不需要区分

如果以上两个条件不满足，那么优化将更加复杂

MySQL 通过外部查询条件，内推到内部查询方式进行优化 IN 查询.如果满足以上两条件将进行直接转换.

针对 IN 与 EXISTS 优化

- 定义表尽量把列声明为 not null
- 如果不需要区分 NULL 与 FLASE 尽量采用以下写法进行优化.

```
outer_expr IN (SELECT inner_expr FROM ...)
```

转换写法如下

```
(outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
```

优化方法只是通过把查询条件内推方式，减少内部查询产生行数。IN 与 EXISTS 操作比较数.外部行数*内部行数.

LIMIT

Limit 是 MySQL 提供分页查询,当 limit 数据越大时候，将越来越慢.

比如针于一个百万级数据 tab_t1

```
SELECT col1,col2,col3 FROM tab_t1 order by id 0,10
SELECT col1,col2,col3 FROM tab_t1 order by id 100000,10
```

第二条语句,将会把的表的前十万条记录全部查询，并且丢弃.只返回 100001,100011 之间数据.浪费 N 多 IO.所以慢.

针对 LIMIT 分页优化就是能过辅助索引进行优化.通过比较辅助索引读取来替代表扫描。避免大量 IO 操作.如果此表包括一 ID 列是主键索引

LIMIT

例-1

针对第二种情况可以改写成以下格式

```
select
  t1.col1,
  t1.col2,
  t1.col3
from tab_t1 t1 ,
(select  t2. id  from tab_t1 t2 order by id limit 100000,10)
where t1.id=t2.id
```

分析查询性能

道先子查询将会覆盖索引扫描.避免从表中直接读取，减小操作 IO 次数来提高查询效率.

然后 t1 表通过索引随机读取 10 条记录

因为索引结构所占用存储远远小于表的所在存储.所以效率得到提升.

LIMIT

例-2

原语句写法

```
select
  s_o_memberid as s_o_memberid ,
  u_gestation_id as u_gestation_id,
  ratio_cld_sex_id as ratio_cld_sex_id,
  ware_ratio_cld_id as ware_ratio_cld_id
from  fact_user_model_kuanbiao
 order by s_o_memberid
 limit 655000 , 1000
```

改进写法

```
select
  t1.s_o_memberid as s_o_memberid ,
  u_gestation_id as u_gestation_id,
  ratio_cld_sex_id as ratio_cld_sex_id,
ware_ratio_cld_id as ware_ratio_cld_id,
from (select
       s_o_memberid from
       fact_user_model_kuanbiao
    order by s_o_memberid limit 655000 , 1000) t2 ,
    fact_user_model_kuanbiao t1
  where t2.s_o_memberid=t1.s_o_memberid
```

修改过后执行时间是 1.9s 左右能返回结果。之前语句执行要 2637s

RAND

功能返回 0-1 之间随机数.在 MySQL 中，根据 order by rand() 进行排序,随机取数.

Rand()影响

- 针对每一行生成随机数.对于大表来说函数调用与行数一致，比较消耗 CPU 资源.
- 根据 order by rand()进行重新排序操作.
-

以上两点对性能影响比较大.当数据量再增加时候很明显会成为性能问题.

优化思路:

预先生成随机数,这个需要解决的问题

首先根据什么生成随机数

再次随机数生成以后该如何使用.

从上面优化思路来看，可以避免排序操作.

在这里提供一种优化方法,通过索引.用recover index 方式生成随机数.然后通过主键与原表进行比较。来优化 order by rand()

RAND

```
mysql> explain select pin from operator_logs order by rand() limit 1 \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: operator_logs
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 109896
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

RAND

```
mysql> explain SELECT
    ->  t1.operator_id
    -> from operator_logs as t1
    -> join (select ceil(rand()*(select max(id) from operator_logs)) as id) as t2
    -> where t1.id>=t2.id
    -> order by t1.id asc
    -> limit 1;
+----+-------------+-----------+-------+---------------+---------+---------+------+-------+--------------------------+
| id | select_type | table     | type  | possible_keys | key     | key_len | ref  | rows  | Extra                    |
+----+-------------+-----------+-------+---------------+---------+---------+------+-------+--------------------------+
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL    | NULL    | NULL |     1 |                          |
|  1 | PRIMARY     | t1        | range | PRIMARY       | PRIMARY | 4       | NULL | 54948 | Using where              |
|  2 | DERIVED     | NULL      | NULL  | NULL          | NULL    | NULL    | NULL |  NULL | No tables used           |
|  3 | SUBQUERY    | NULL      | NULL  | NULL          | NULL    | NULL    | NULL |  NULL | Select tables optimized away |
+----+-------------+-----------+-------+---------------+---------+---------+------+-------+--------------------------+
```

Order 、 Group 、 Distinct 、 Count

Order

Order 是一种排序操作，需要内存与 CPU 消耗较多.针对 Order 排序操作优化思路就是
通过有序结构，进行数据读取，而避免排序操作。而有序操作就是索引.(b-tree)

```
SELECT col1,col2,col3 from where col3=constant order by col1

Idx_1 (col3,col1)
```

如果存在 index idx_1 那么上条语句，排序操作就会消除.

Order 、 Group 、 Distinct 、 Count

Order

但是在以下情况通过索引是消除不掉排序操作的.

```
order by col1,col2
idx_1(col1)
idx_2(col2)
```

通过不同索引键进行排序操作.

```
order by col1 desc,col2 asc
idx_1(col1,col2)
```

混合 desc 与 asc 的使用

```
Order by col1, group by col3
Idx_1(col1,col3)
```

Order by 与 group by 使用不同表达式

Order 、 Group 、 Distinct 、 Count

Group

Group 分组优化通过索引优化，有两种索引扫描方式

松散索引方式 loose index scan

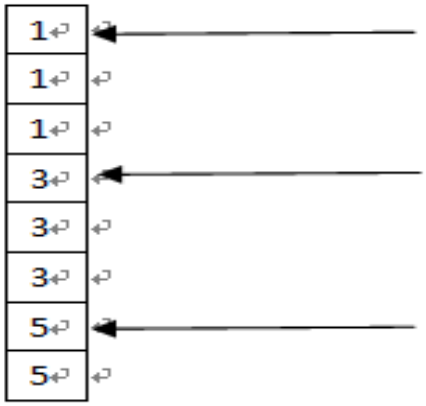
紧凑索引方式 Tight Index Scan

松散索引方式

MySQL 通过有序索引结构进行分组排序，而不需要读取所有键值.称之松散索引方式.

适合条件

- 针对单张数据表分组查询
- 聚合函数支持 max()、min()
- 不适用前缀索引结构
- 查询列全部基于索引满足覆盖索引条件



当出现松散索引扫描通过 explain 在 extra 信息显示: Using index for group-by.

松散索引扫描方式，比覆盖索引更加有效.

Order 、 Group 、 Distinct 、 Count

Group

紧凑索引方式

这种扫描方式，依据查询条件可以全索引扫描或是范围索引扫描

Table:T1

Index:idx_1 (c1,c2,c3,c4)

以下两种情况依然出现紧凑索引扫描方式

- 分组有间隙，但 c2 是常量

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- 没有依据索引第一列，但是第一列以常量形式出现在 where 条件

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

Order 、 Group 、 Distinct 、 Count

Group

```
mysql> explain select id  from  operator_logs group by id \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: operator_logs
         type: range
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 109210
        Extra: Using index for group-by

mysql> explain select activity_id from award_user group by type \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: award_user
         type: index
possible_keys: NULL
          key: idx_activityid_type
      key_len: 5
          ref: NULL
         rows: 5226
        Extra: Using index; Using temporary; Using filesort

mysql> explain select activity_id from award_user where activity_id=10000369 group by type \G;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: award_user
         type: ref
possible_keys: idx_activityid_type
          key: idx_activityid_type
      key_len: 4
          ref: const
         rows: 3
        Extra: Using where; Using index
```

Order 、 Group 、 Distinct 、 Count

Distinct/Count

- **Distinct**

Distinct 优化与 group by 类似，能应用于 group by 优化同样适合于 distinct 优化.

如

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

这两个查询是相同的.所以 group by 优化适用场景同样适用 distinct 场景.

- **Count**

在实际计算过程，如果不是对于空值不敏感查询语句尽量使用 count(*).
对于 count 操作在优化层次，应该优化 covering index

HJ、SMJ、NL

算法类型	算法描述
SORT-MERGE JOIN	1.对于 join 字段进行排序.然后进行合并 2.排序代价昂贵
NESTED LOOP	1.驱动表检索数据与内源表进行比较输出 2.对于驱动表较大时间，操作代价较高
HASH JOIN	1.关联小表在内存创建 hash table 2.大表创建行源，作为探测输入表源 3.hash join 基于 CBO.

HJ、SMJ、NL

NESTED LOOP

1. 从 outer table 读取一行记录
2. 读取记录与 inner table 进行比较

优化两点:

outer table 结果集较小,就是行源数较少.

与 inner table 进行较是通过 index 进行.

例

table	Join type
T1	rang
T2	ref
T3	all

```
for each row in (SELECT col1,col2 from t1 where t1.col1>5 and t1.col1<10)
{for each row in (select col1,col2 from t2 where t1.col1=t2.col2)
    {for each row in (select col1,col2 from t3 where t3.col2=t1.col2)
      {if row satisfies join conditions,
          send to client
      }
    }
}
```

Sql Summary

- 参数类型
- 书写过程避免dead code
- 字符集影响

Dead Code

不必要括号

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

恒等式

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

冗余条件

```
 (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

Sql Summary

参数类型

通过where通过传入类型应该与列类型致

- 造成执行计划不稳定
- 发生隐式类型转换操作
- 增加额外资源消耗

字符集

多表Join操作，字符集要一致

- 执行计划不稳定,造成错误的执行计划

谢谢

zhangyuanxiang@360buy.com