



EXPLAIN TYPE 列的 JOIN 常见场景详解（上）

爱可生开源社区 2025-01-06 35 阅读6分钟

专栏连载至此，相信读者们已经对一条 SQL 的优化步骤、执行计划等有了一个大体的了解。那接下来我们对 MySQL 的执行计划输出进行详细解释，以便大家对其了解的更加深入。

作者：杨涛涛，爱可生技术专家。

爱可生开源社区出品，原创内容未经授权不得随意使用，转载请联系小编并注明来源。

我们这个标题为什么叫做EXPLAIN TYPE 列的JOIN 常见场景详解呢？从MySQL 优化器的角度来看，所有SQL都是JOIN查询（单表检索可以看成过滤字段和主键做JOIN的特殊类型）。由于内容太多，我分成了上下两部分，今天我们来从第一部分开始。

还是表t1，不过我对表结构做了少许变更，更改原来的自增主键为联合主键(f0,f1)，表记录数不变，还是10W行。

mysql

[代码解读](#)[复制代码](#)

```
1 CREATE TABLE `t1` (  
2   `f0` int NOT NULL,  
3   `f1` int NOT NULL,  
4   `r1` int DEFAULT NULL,  
5   `r2` int DEFAULT NULL,  
6   `r3` int DEFAULT NULL,  
7   `log_date` date DEFAULT NULL,  
8   PRIMARY KEY (`f0`,`f1`)  
9 ) ENGINE=InnoDB
```

接下来，我写了几条简单的SQL，来分别讲讲type列的意义。

第一， type 栏为"const"

这表明排除索引性能的话，这条SQL 一定是最优的。比如 SQL 1： 过滤字段为联合主键，并且是两个固定的常量比对，这种一定是最优化的：

SQL 1： select * from t1 where f0=110 and f1 = 778

执行计划如下： type 栏里是“const”， ref 栏里是const,const。表明扫描表t1，给定两个常量来过滤，同时走的索引是主键，可以联合rows栏一起看，如果type栏相同，那么rows栏数值小的肯定较为优化。

mysql

[代码解读](#)[复制代码](#)

```
1  debian-ytt1:ytt>desc select * from t1 where f0=110 and f1 = 778\G  
2  ***** 1. row *****  
3      id: 1  
4      select_type: SIMPLE  
5      table: t1  
6      partitions: NULL  
7      type: const  
8      possible_keys: PRIMARY  
9      key: PRIMARY  
10     key_len: 8  
11     ref: const,const  
12     rows: 1  
13     filtered: 100.00  
14     Extra: NULL  
15  1 row in set, 1 warning (0.00 sec)
```

第二， type 栏为"eq_ref"

这其实和const类似，也是优化比率靠前的，不同的是eq_ref用于两张真实的表JOIN，并且两表的JOIN KEY 必须为主键（或者唯一索引）的全部，同时对于被驱动表而言，对它进行检索的过滤条件是驱动表的所有主键，每次只有一行。（关于JOIN 的优化我会另外开篇细讲，这里就不多说了）

比如**SQL 2： select * from t1 join t2 using(f0,f1)**

SQL 2 是两表做内联，并且联接的键为两表的主键，这样的SQL 语句（仅从SQL 语句角度，不掺杂业务逻辑）是两表联接类型里不带过滤条件的场景下最优的。

那依然看下执行计划： 这里省去表t2的执行计划，只看表t1。对于表t1来讲，对它的扫描基于主键，并且在扫描主键时，每次给的常量值为表t2的联合主键，而且是非常精确的一行。

 爱可生开源社区

LV.5

一个有深度的数据库社区 @上...

作者榜No.8

优秀作者

224

文章

237k

阅读

170

粉丝

关注

私信

目录 收起

第一， type 栏为"const"

第二， type 栏为"eq_ref"

第三， type 栏为"ref"

第四， type 栏为"range"

第五， type 栏为"index"

关于 SQLE

- 相关推荐
- 列举GaussDB(DWS)常见的查询时索引...

473阅读 · 1点赞
- Mysql Explain之type详解

19k阅读 · 25点赞
- Hive的执行计划（Explain）

995阅读 · 2点赞
- 什么是MySQL的执行计划（Explain关键...

6.8k阅读 · 63点赞
- 深入浅出Mysql（二） Explain详解

1.1k阅读 · 10点赞

- 精选内容
- AWS 弹性伸缩特性介绍

AutoMQ · 20阅读 · 0点赞
- 鸿蒙轻内核A核源码分析系列六 MMU协...

别说我什么都不会 · 11阅读 · 0点赞
- OpenHarmony（鸿蒙南向开发）——轻...

塞尔维亚大汉 · 19阅读 · 0点赞
- Java 虚拟线程（Virtual Threads）

魔镜前的帅比 · 65阅读 · 0点赞
- MySQL索引优化-Index Condition Push...

喵喵帕斯 · 48阅读 · 2点赞

找对属于你的技术圈子

回复「进群」加入官方微信群





mysql

代码解读

复制代码

```
1  debian-ytt1:ytt>desc select * from t1 join t2 using(f0,f1)\G
2  ...
3  ***** 2. row *****
4      id: 1
5      select_type: SIMPLE
6      table: t1
7      partitions: NULL
8      type: eq_ref
9  possible_keys: PRIMARY
10     key: PRIMARY
11    key_len: 8
12     ref: ytt.t2.f0,ytt.t2.f1
13    rows: 1
14   filtered: 100.00
15     Extra: NULL
16  2 rows in set, 1 warning (0.00 sec)
```

第三，type 栏为"ref"

ref 和eq_ref 类似，不同的是两表的JOIN KEY 非主键、非唯一索引。这种场景从SQL角度来讲，应该避免掉；如果实在无法避免，可以想办法减少两表JOIN的记录数。

那对SQL2 做些调整，变为SQL 3： JOIN 条件变为字段r1，并且同时给两表字段r1加索引。

SQL 3： **select * from t1 a join t2 b using(r1)**

再看下查询计划：还是省去表t1，只看表t2的执行计划。这里对表t2的检索走索引idx_r1，同时每次扫描引用表t1字段r1，可以结合rows栏来看，这条SQL其实并不优化。

mysql

代码解读

复制代码

```
1  debian-ytt1:ytt>desc select * from t1 a join t2 b using(r1)\G
2  ...
3  ***** 2. row *****
4      id: 1
5      select_type: SIMPLE
6      table: b
7      partitions: NULL
8      type: ref
9  possible_keys: idx_r1
10     key: idx_r1
11    key_len: 5
12     ref: ytt.a.r1
13    rows: 19838
14   filtered: 100.00
15     Extra: NULL
16  2 rows in set, 1 warning (0.01 sec)
17
18
```

第四，type 栏为"range"

range 代表范围扫描，和前面三个不同，前面三个都是基于常量。

来看下SQL 4： **select * from t1 where f0<120**

SQL 4 对表t1的检索条件是一个范围（-INF,120），执行计划如下： 对表t1的扫描走主键，类型为range。

mysql

代码解读

复制代码

```
1  debian-ytt1:ytt>desc select * from t1 where f0<120\G
2  ***** 1. row *****
3      id: 1
4      select_type: SIMPLE
5      table: t1
6      partitions: NULL
7      type: range
8  possible_keys: PRIMARY
9      key: PRIMARY
10     key_len: 4
11     ref: NULL
12    rows: 93
13   filtered: 100.00
14     Extra: Using where
15  1 row in set, 1 warning (0.00 sec)
```

SQL 4 是对表t1的范围扫描，有些时候基于一些表记录特殊性（不具备通用性），可以把范围扫描优化为常量扫描。这里表t1就具有特殊性，对于字段过滤条件为f0<120的结果和过滤条件为f0=110的结果是一样的，所以改SQL 4 为 SQL 5：

SQL 5： **select * from t1 where f0=110**

看下SQL 5的执行计划： 成功把对表t1的范围扫描变为常量扫描，type 栏由range 变为ref.

mysql

[代码解读](#)[复制代码](#)

```
1  debian-ytt1:ytt>desc select * from t1 where f0=110\G
2  ***** 1. row *****
3      id: 1
4      select_type: SIMPLE
5      table: t1
6      partitions: NULL
7      type: ref
8      possible_keys: PRIMARY
9      key: PRIMARY
10     key_len: 4
11     ref: const
12     rows: 93
13     filtered: 100.00
14     Extra: NULL
15  1 row in set, 1 warning (0.00 sec)
```

其实这点从传统的执行计划结果里看不出什么效果，还是得实际执行后，看两条SQL 的执行成本。我们使用explain analyze 来对比下SQL 4和 SQL 5的执行成本： SQL 4 成本为18.93， SQL 5成本为 9.62， 性能提升很明显。

mysql

[代码解读](#)[复制代码](#)

```
1  debian-ytt1:ytt>desc analyze select * from t1 where f0< 120\G
2  ***** 1. row *****
3  EXPLAIN: -> Filter: (t1.f0 < 120) (cost=18.93 rows=93) (actual time=0.040..0.061 rows=93 loops=1)
4      -> Index range scan on t1 using PRIMARY (cost=18.93 rows=93) (actual time=0.038..0.047 rows=93)
5
6  1 row in set (0.00 sec)
7
8  debian-ytt1:ytt>desc analyze select * from t1 where f0=110\G
9  ***** 1. row *****
10 EXPLAIN: -> Index lookup on t1 using PRIMARY (f0=110) (cost=9.62 rows=93) (actual time=0.065..0.087 rows=93)
11
12  1 row in set (0.00 sec)
```

第五， type 栏为"index"

Index 表示覆盖索引扫描，可以简单描述为没有过滤条件的索引扫描；更进一步，如果从索引角度来讲，就是全表扫了。

比如SQL 6: **select r1 from t1 limit 10**

SQL 6 扫描的列只有r1，而非全部字段，此刻走索引idx_r1即可，不需要回表。

执行计划如下： type 为Index, 使用索引idx_r1, 扫描行数为10W行，刚好表t1总记录数也是10W.

mysql

[代码解读](#)[复制代码](#)

```
1  debian-ytt1:ytt>desc select r1 from t1 limit 10 \G
2  ***** 1. row *****
3      id: 1
4      select_type: SIMPLE
5      table: t1
6      partitions: NULL
7      type: index
8      possible_keys: NULL
9      key: idx_r1
10     key_len: 5
11     ref: NULL
12     rows: 106313
13     filtered: 100.00
14     Extra: Using index
15  1 row in set, 1 warning (0.00 sec)
```

其实对于SQL 6 来讲，有limit 10 子句是可以提前终止扫描的，但是这里MySQL为什么还是扫描所有行？ 这里MySQL虽然走了索引idx_r1,但是没有排序子句，进而造成MySQL 不知道按照什么顺序输出，只能扫描所有记录。

对于这类的优化，可以加一个排序子句，把现有索引的预排序特性利用上，变为 SQL 7:

SQL 7: select r1 from t1 order by r1 limit 10;

此时再查看查询计划：很显然，MySQL根据利用索引idx_r1的有序性，加上limit 子句，提前终止了扫描。

mysql

[代码解读](#)[复制代码](#)

```
1  debian-ytt1:ytt>explain select r1  from t1 order by r1 limit 10\G
2  ***** 1. row *****
3      id: 1
4      select_type: SIMPLE
5      table: t1
6      partitions: NULL
7      type: index
8      possible_keys: NULL
9      key: idx_r1
10     key_len: 5
11     ref: NULL
12     rows: 10
13     filtered: 100.00
14     Extra: Using index
15  1 row in set, 1 warning (0.00 sec)
```

关于EXPLAIN TYPE 栏 的JOIN常见场景上篇就到这里了，欢迎大家订阅下一篇。


更多技术文章，请访问：opensource.actionsky.com/

关于 SQLE

SQLE 是一款全方位的 SQL 质量管理平台，覆盖开发至生产环境的 SQL 审核和管理。支持主流的开源、商业、国产数据库，为开发和运维提供流程自动化能力，提升上线效率，提高数据质量。

标签：[数据库](#)

评论 0



登录 / 注册

即可发布评论!



暂无评论数据

为你推荐

35 张图带你 MySQL 调优

程序员cxuan | 3年前 | 4.8k | 57 | 2 [后端](#) [MySQL](#)

带你看懂MySQL执行计划

MySQL技术 | 3年前 | 1.5k | 3 | 评论 [MySQL](#)

MySQL SQL的完整处理流程

终有救赎 | 1年前 | 378 | 11 | 评论 [后端](#) [面试](#) [数据库](#)

MySQL的SQL执行计划分析及【关键指标】讲解

大浪 | 4月前 | 149 | 1 | 评论 [数据库](#)

拜托别再问我MySQL性能如何优化了？这篇送你！！

码猿技术专栏 | 4年前 | 3.4k | 7 | 1 [MySQL](#)

SQL性能优化技巧

橘子coding | 3年前 | 2.3k | 21 | 评论 [MySQL](#) [后端](#)

MySQL优化方案

yangnk | 1年前 | 472 | 3 | 评论 [后端](#)

SQL优化13连问，收藏好

捡田螺的小男孩 | 1年前 | 14k | 141 | 9 [面试](#) [数据库](#) [Java](#)

MySQL索引（六）索引优化补充，分页查询、多表查询、统计查询

鳄鱼儿 | 10月前 | 471 | 5 | 评论 [数据库](#) [MySQL](#) [搜索引擎](#)

EXPLAIN：解说一条简单 SQL 语句的执行计划

爱可生开源社区 | 2月前 | 70 | 点赞 | 评论 [数据库](#)

从执行计划了解MySQL优化策略

终有救赎 | 1年前 | 835 | 2 | 评论 [后端](#) [面试](#) [Java](#)

【SQL性能提升篇🔥】MySQL常用优化工具Explain、Trace工具、优化案例

JingYu | 11月前 | 916 | 14 | 评论 [后端](#)

show processlist 命令详解，MySQL优化看这一篇就够了

公众号_IT老哥 | 4年前 | 7.8k | 12 | 评论 [MySQL](#)

MySQL源码解析之执行计划

GreatSQL | 2年前 | 570 | 2 | 评论 [数据库](#)

MySql优化-上|8月更文挑战

耶马 | 3年前 | 226 | 1 | 评论 [MySQL](#) [后端](#)