不会看EXPALIN执行计划，算什么会SQL优化?

原创 胖虎 Java后端精选 4天前

超全面！Java核心知识总结（点击查看）

超全面！Java核心知识总结（点击查看）

我们平时使用的关系型数据库MySQL中，查询优化器是非常核心的一个组件。它决定对特定的查询使用哪些索引、哪些关联算法。从而更效率的去执行SQL语句。

而 `EXPLAIN` 命令就是查看查询优化器如何决定执行查询的主要方法。

使用 `EXPLAIN` 的方法也很简单，只需要在 `select` 关键词前加 `explain` 即可。

```
explain select * from user;
```

执行上面的sql，并不会返回执行的结果，返回的是执行计划中每一步的信息。

> 本文使用的MySQL版本为：8.0.18

EXPLAIN的作用

官方对于EXPLAIN的定义

```
The EXPLAIN statement provides information about how MySQL executes statements. EXPLAIN works
```

模拟优化器是如何执行select语句，让我们知道效率低下的原因。从而改进sql语句的执行效率。

看下面一个很简单的例子

```
mysql> EXPLAIN SELECT * FROM pms_product where id=1;
+----+-------------+-------------+------------+-------+---------------+---------+---------+---
| id | select_type | table       | partitions | type  | possible_keys | key     | key_len | re
+----+-------------+-------------+------------+-------+---------------+---------+---------+---
```

```
|  1 | SIMPLE      | pms_product | NULL       | const | PRIMARY       | PRIMARY | 8       | cc
+----+-------------+-------------+------------+-------+---------------+---------+---------+---

1 row in set, 1 warning (0.00 sec)
```

从上面的执行结果，可以看出 `EXPLAIN` 的执行结果有以下几个字段参数

id 查询序列号

用来标识SQL执行顺序的关键词，可以理解为优先级。

按照从大到小的顺序执行。

- 如果多条执行计划的id相同，则从上到下顺序执行。
- 如果有子查询，id的序号会增加。序号越大代表优先级越高，越先被执行

```
mysql> EXPLAIN SELECT id from oms_order where oms_order.id = (select order_id from oms_order_i
+----+-------------+----------------+------------+-------+---------------+---------+---------+
| id | select_type | table          | partitions | type  | possible_keys | key     | key_len |
+----+-------------+----------------+------------+-------+---------------+---------+---------+
|  1 | PRIMARY     | oms_order      | NULL       | const | PRIMARY       | PRIMARY | 8       |
|  2 | SUBQUERY    | oms_order_item | NULL       | ALL   | NULL          | NULL    | NULL    |
+----+-------------+----------------+------------+-------+---------------+---------+---------+

2 rows in set, 1 warning (0.00 sec)
```

- 有多个相同的id，可以认为是一组查询。自上往下顺序执行。

```
mysql> EXPLAIN SELECT oi.id item_id,
    ->             oi.product_name item_product_name
    ->         FROM
    ->             oms_order o
    ->             LEFT JOIN oms_order_item oi ON o.id = oi.order_id
    ->             LEFT JOIN oms_order_operate_history oh ON o.id = oh.order_id
    ->         WHERE
    ->             o.id = 12;

+----+-------------+-------+------------+-------+---------------+---------+---------+-------+-
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+-
|  1 | SIMPLE      | o     | NULL       | const | PRIMARY       | PRIMARY | 8       | const |
|  1 | SIMPLE      | oi    | NULL       | ALL   | NULL          | NULL    | NULL    | NULL  |
```

```
|  1 | SIMPLE      | oh    | NULL       | ALL   | NULL          | NULL    | NULL    | NULL  |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+-
3 rows in set, 1 warning (0.00 sec)
```

select_type：查询类型

表示当前查询类型中select语句的类型，分为以下几类

- SIMPLE：简单的SELECT查询，可有理解为最基本的查询，没有任何子查询、UNION等
- PRIMARY：查询中最外层的SELECT
- SUBQUERY：子查询的第一个SELECT，不依赖外部查询
- DEPENDENT SUBQUERY：子查询中的第一个SELECT，依赖外部查询。
- UNION：UNION查询中处于内层的SELECT（内层的SELECT语句与外层的SELECT语句没有依赖关系）
- DEPENDENT UNION：UNION中的第二个或后面的SELECT语句，取决于外面的查询)
- UNION RESULT：UNION操作的结果，id一般为null
- DERIVED：派生表的SELECT，例如： `select aa from （select * from a） b`
- UNCACHEABLE SUBQUERY：对于外层的主表，子查询不可被物化，每次都需要计算（耗时）

table：数据库名

表示当前执行计划中所访问的数据库中的表名，如果有别名则显示别名。

type：访问类型

不同版本的数据库 `type` 有点不同，在MySQL5.7 `type` 的方式高达14种！这里只说一下常见的六种类型，访问效率依次增强。

- ALL：遍历全表找到匹配的行，也就是我们常说的"全表查询"
- indexs：同样是扫描全表，区别在于扫描顺序是按照索引的顺序去扫描
- range：范围内的索引扫描，比如常用的关键词 `between` 、 `>` 、 `<` 、 `in` 、 `or` 等.
- ref：查询条件是使用了不是主键且不是唯一的索引。当查询到某一条数据，虽然会进行小范围的继续查找，但不至于全表扫描
- eq_ref：与 `ref` 相比，区别就是使用了唯一索引
- const：主键放到 `where` 后面作为条件查询，MySQL优化器会讲这次查询优化成一个常量，如何转化取决于自身的优化器。

possible_keys：显示索引

显示可能应用在这张表中的索引，可能是一个或者多个。查询涉及到的字段上若存在索引，则该索引会被列出，但是查询中不一定会使用。

如果没有任何索引则显示NULL。

Key：实际索引

显示查询中实际使用的索引，该索引已经存在于 `possible_keys` 中。如果没有索引，显示NULL，如果想强制MySQL使用或忽视possible_keys列中的索引，在查询中使用force index、ignore index。

key_len：索引预估长度

显示的值为索引的最大可能长度/使用的字节数（并不是真实长度,是根据表定义计算而得来的），在不损失精确性的情况下， `key_len` 的值越小越好

ref：实际使用的索引

主要是记录了在 `Key` 列中记录的索引 ，进行表查找时所用到的列或者常量。常见的比如 `const(常量)`

一般是查询条件或关联条件中等号右边的值，如果是常量那么ref列是const，非常量的话ref列就是字段名。

rows：预估行数

现实的内容为，执行SQL语句，预估要读取并检测的行数（并不是结果的行数）

filtered

在MySQL 5.7版本之前如果要显示 `filtered` 需要使用 `explain extended` 命令，而在5.7之后，直接使用 `explain` 就会默认显示该字段了。

字段对应的值是个百分比，表示表里符合条件的记录数的百分比。就是存储引擎返回的数据在server层过滤后，剩下多少满足查询的记录数量的比例，注意是百分比，不是具体记录数。

Extra：额外信息说明

常见的有

- Using where: 查询条件中
- Using index condition: 索引下推（Index Condition Pushdown）
- Using filesort：Server层需要做额外的排序操作，需要优化，让排序使用到索引
- no matching row in const table：唯一索引（包括主键）查询不到数据
- Using index：覆盖索引，查询的行都在对应的索引中
- Using MRR：MRR优化，就是为了减少访问磁盘的次数，回表时，先把主键索引排序后再访问。因为接近的索引，可能在相同的页上
- Using temporary：使用了临时表，需要优化

总结