> 作者：千锋-索尔
>
> 版本：QF1.0
>
> 版权：千锋Java教研院

# Explain介绍

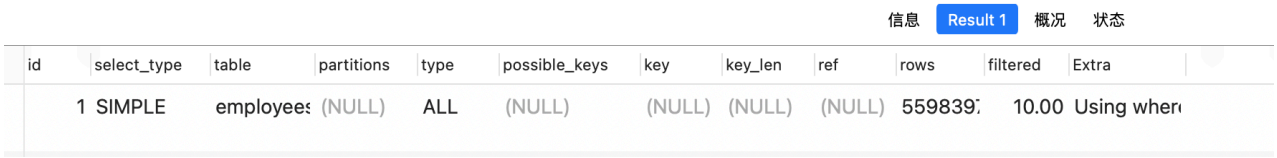
## 1.Explain工具的介绍

MySQL自带的一块用于查看SQL性能的工具，通过该工具显现出的提示信息，可以知道当前SQL语句的执行性能，及慢的原因。

使用Explain工具：

```
EXPLAIN select * from employees where age=20
```

看到如下的内容，判断出该SQL语句的执行性能及慢的原因




## 2.MySQL的内部优化器

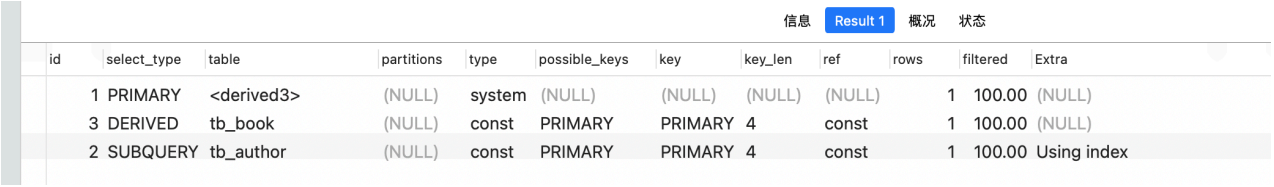
MySQL提供了一个优化器，该优化器会对你的sql语句、索引的选择等等做出一些优化，让执行的性能会更好。

```
EXPLAIN select * from tb_book where id=1;
show WARNINGS;
=======>显示的内容：
* select#1 */
EXPLAIN select '1' AS `id`,'qf' AS `name` from `db_mysql_pro`.`tb_book` where
true
```

## 3.Explain中的select_type列

- 简单的查询类型：simple
- 复杂的查询：

```
# 关闭衍射表合并
set session optimizer_switch='derived_merge=off';
# 复杂查询
EXPLAIN SELECT ( SELECT 1 from tb_author where id=1) from (SELECT * from
tb_book WHERE id=1) der;
```



信息 Result 1 概况 状态

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | PRIMARY | <derived3> | (NULL) | system | (NULL) | (NULL) | (NULL) | (NULL) | 1 | 100.00 | (NULL) |
| 3 | DERIVED | tb_book | (NULL) | const | PRIMARY | PRIMARY | 4 | const | 1 | 100.00 | (NULL) |
| 2 | SUBQUERY | tb_author | (NULL) | const | PRIMARY | PRIMARY | 4 | const | 1 | 100.00 | Using index |

- derived：在from之后的子查询：产生的结果会存入到衍生表（临时）中
- subquery：在from之前，select之后的子查询
- primary：最外部的主查询
- UNION：联合查询

# 4.id列

id越大的越先执行，id相同的，谁在上面谁先执行

# 5.type列

type列直接描述了当前sql的性能——有没有走索引。type列的各值的性能排序：

```
NULL > system > const > eq_ref > ref > range > index > ALL
```

尽量保证sql的性能是在index之上的。

- NULL

```
EXPLAIN select min(id) from tb_book
```

这一次查询直接从索引树上就能获得结果。

- system

```
EXPLAIN SELECT ( SELECT 1 from tb_author where id=1) from (SELECT * from
tb_book WHERE id=1) der;
```

最外部的select查询的时候，要查询的数据只有1条，那么直接拿出来使用即可，因此性能非常好，是system，但是这种场景很少见。

- const

用主键和一个常量进行等值的比较，性能非常强

```
EXPLAIN SELECT * from tb_book where id=1
```

- eq_ref

在join查询中，被关联的表使用了主键作为关联条件，那么类型是eq_ref

```
EXPLAIN SELECT * from tb_book_author left join tb_book on
tb_book_author.book_id=tb_book.id
```

信息 Result 1 概况 状态

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | tb_book_author | (NULL) | ALL | (NULL) | (NULL) | (NULL) | (NULL) | 3 | 100.00 | (NULL) |
| 1 | SIMPLE | tb_book | (NULL) | eq_ref | PRIMARY | PRIMARY | 4 | db_mys | 1 | 100.00 | (NULL) |

- ref
  - 简单查询: 在简单查询中，使用了非主键索引做等值匹配

```
EXPLAIN select * from tb_book where name = 'a'
```

信息 Result 1 概况 状态

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | tb_book | (NULL) | ref | idx_name | idx_name | 33 | const | 1 | 100.00 | Using index |

  - 复杂查询:被关联查询的表使用了非主键索引作为关联条件

```
EXPLAIN SELECT book_id from tb_book  left join tb_book_author  on
tb_book_author.book_id=tb_book.id
```

信息 Result 1 概况 状态

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | tb_book | (NULL) | index | (NULL) | idx_name | 33 | (NULL) | 19 | 100.00 | Using index |
| 1 | SIMPLE | tb_book_author | (NULL) | ref | idx_book_id_a | idx_book_ | 4 | db_mys | 1 | 100.00 | Using index |

- range

  使用主键索引做范围查询

```
EXPLAIN SELECT * from tb_book where id>1
```

- index

  使用了索引列做查询，覆盖索引也是这种结果（也是使用了索引列）

```
-- 使用了索引列做查询
EXPLAIN select * from tb_book where name > 'a'
-- ID name  使用了覆盖索引
EXPLAIN select * from tb_book
-- 使用了覆盖索引
EXPLAIN select id from tb_author
```

什么是覆盖索引：要查询的列，都是索引列

- all

  全表扫描。

# 6.possible-keys

可能使用到的索引列

# 7.key

实际使用到的索引列，注意：不一定说可能使用到的索引列一定就是实际使用到的索引列。

# 8.key-len

实际使用的索引列的长度，在组合索引中得知具体用了哪些索引列

```
EXPLAIN SELECT * FROM employees where name='abc' and position='dev'
```

信息 Result 1 概况 状态

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | employees | (NULL) | ref | idx_name_age_position | idx_name_age_position | 74 | const | 1 | 10.00 | Using index condition |

Key-len的计算规则：

```
EXPLAIN select * from employees where name = 'customer10011' and age=30 and
position='dev'
- 字符串
1. char(n): n字节长度
2. varchar(n): 2字节存储字符串长度,如果是utf-8,则长度3n + 2

- 数值类型
1. tinyint: 1字节
2. smallint: 2字节
3. int: 4字节
4. bigint: 8字节

- 时间类型
1. date: 3字节
```

```
2. timestamp: 4字节
3. datetime: 8字节

如果字段允许为NULL,需要1字节记录是否为NULL
索引最大长度是768字节,当字符串过长时, mysql会做一个类似左前缀索引的处理,将前半部分的字符提取出来做索引。
```

# 9.rows

可能要查询行数的近似值。

MySQL中数据的单位都是页，MySQL又采用了采样统计的方法，采样统计的时候，InnoDB默认会选择N个数据页，统计这些页面上的不同值，得到一个平均值，然后乘以这个索引的页面数，就得到了这个索引的基数。

# 10.extra

这一次查询给到的额外的信息

- using index: 使用了覆盖索引

```
EXPLAIN select id,name from tb_author
```

- using where: 没有使用索引，做条件查询 (需要优化)

```
EXPLAIN select * from tb_book where name > 'a'
```

- using index condition：没有覆盖索引，但查询条件中使用了索引列

```
EXPLAIN SELECT * FROM employees where name='abc' and position='dev'
```

- Using temporary： 需要创建临时表来完成操作，性能较差。建议优化。

```
EXPLAIN select DISTINCT name from tb_book
```

- Using filesort: 使用了文件排序，文件排序实际上也有区别，在之后的文件排序的原理中再介绍

```
EXPLAIN select *  from tb_book order by name
```

- Select tables optimized away

使用了聚合函数操作索引列

```
EXPLAIN select min(id) from tb_book
```