explain分析执行计划

可以通过 EXPLAIN获取 MySQL如何执行 SELECT 语句的信息，包括在 SELECT 语句执行过程中表如何连接和连接的顺序。

explain sql语句

字段	含义
id	select查询的序列号，是一组数字，表示的是查询中执行select子句或者是操作表的顺序。
select_type	表示 SELECT 的类型，常见的取值有 SIMPLE（简单表，即不使用表连接或者子查询）、PRIMARY（主查询，即外层的查询）、UNION（UNION 中的第二个或者后面的查询语句）、SUBQUERY（子查询中的第一个 SELECT）等
table	输出结果集的表
type	表示表的连接类型，性能由好到差的连接类型为(system ---> const -----> eq_ref ------> ref -------> ref_or_null----> index_merge ---> index_subquery -----> range -----> index ------> all)
possible_keys	表示查询时，可能使用的索引
key	表示实际使用的索引
key_len	索引字段的长度
rows	扫描行的数量
extra	执行情况的说明和描述

数据准备

```
CREATE TABLE `t_role` (
    `id` varchar(32) NOT NULL,
    `role_name` varchar(255) DEFAULT NULL,
    `role_code` varchar(255) DEFAULT NULL,
    `description` varchar(255) DEFAULT NULL,
     PRIMARY KEY (`id`),
     UNIQUE KEY `unique_role_name` (`role_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `t_user` (
    `id` varchar(32) NOT NULL,
    `username` varchar(45) NOT NULL,
    `password` varchar(96) NOT NULL,
    `name` varchar(45) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY `unique_user_username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `user_role` (
    `id` int(11) NOT NULL auto_increment ,
    `user_id` varchar(32) DEFAULT NULL,
    `role_id` varchar(32) DEFAULT NULL,
    PRIMARY KEY (`id`),
    KEY `fk_ur_user_id` (`user_id`),
    KEY `fk_ur_role_id` (`role_id`),
```

```
    CONSTRAINT `fk_ur_role_id` FOREIGN KEY (`role_id`) REFERENCES `t_role`
(`id`) ON
    DELETE NO ACTION ON UPDATE NO ACTION,
    CONSTRAINT `fk_ur_user_id` FOREIGN KEY (`user_id`) REFERENCES `t_user`
(`id`) ON
    DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
insert into `t_user` (`id`, `username`, `password`, `name`)
values('1','super','$2a$10$TJ4TmCdK.X4wv/tCqHW14.w7OU3CC33CeVncD3SLmyMXMknstqKRe
','超级管理员');
insert into `t_user` (`id`, `username`, `password`, `name`)
values('2','admin','$2a$10$TJ4TmCdK.X4wv/tCqHW14.w7OU3CC33CeVncD3SLmyMXMknstqKRe
','系统管理员');
insert into `t_user` (`id`, `username`, `password`, `name`)
values('3','qf','$2a$10$8qmaHgUFUAmPR5pOuWhYWOr291WJYjHelUlYnO7k5ELF8ZCrWOCui',
'test02');
insert into `t_user` (`id`, `username`, `password`, `name`)
values('4','stu1','$2a$10$pLtt2KDAFpwTWLjNsmTEi.oU1yOZyIn9XkziK/y/spH5rftCpUMZa'
,'学生1');
insert into `t_user` (`id`, `username`, `password`, `name`)
values('5','stu2','$2a$10$nxPKkYSez7uz2YQYUnwhR.z57km3yqKn3Hr/p1FR6ZKgc18u.Tvqm'
,'学生2');
insert into `t_user` (`id`, `username`, `password`, `name`)
values('6','t1','$2a$10$TJ4TmCdK.X4wv/tCqHW14.w7OU3CC33CeVncD3SLmyMXMknstqKRe','
老师1');

INSERT INTO `t_role` (`id`, `role_name`, `role_code`, `description`)
VALUES('5','学生','student','学生');
INSERT INTO `t_role` (`id`, `role_name`, `role_code`, `description`)
VALUES('7','老师','teacher','老师');
INSERT INTO `t_role` (`id`, `role_name`, `role_code`, `description`)
VALUES('8','教学管理员','teachmanager','教学管理员');
INSERT INTO `t_role` (`id`, `role_name`, `role_code`, `description`)
VALUES('9','管理员','admin','管理员');
INSERT INTO `t_role` (`id`, `role_name`, `role_code`, `description`)
VALUES('10','超级管理员','super','超级管理员');

INSERT INTO user_role(id,user_id,role_id) VALUES(NULL, '1', '5'),(NULL, '1',
'7'), (NULL, '2', '8'),(NULL, '3', '9'),(NULL, '4', '8'),(NULL, '5', '10') ;
```

3.3.2 explain 之 id

id 字段是 select查询的序列号，是一组数字，表示的是查询中执行select子句或者是操作表的顺序。id 情况有三种:
1) id 相同表示加载表的顺序是从上到下。

```
explain select * from t_role r, t_user u, user_role ur where r.id = ur.role_id
and u.id = ur.user_id ;
```

2) id 不同id值越大，优先级越高，越先被执行。

```
EXPLAIN SELECT * FROM t_role WHERE id = (SELECT role_id FROM user_role WHERE
user_id = (SELECT id FROM t_user WHERE username = 'stu1'))
```

3) id 有相同，也有不同，同时存在。id相同的可以认为是一组，从上往下顺序执行;在所有的组中，id的值越 大，优先级越高，越先执行。

```
EXPLAIN SELECT * FROM t_role r , (SELECT * FROM user_role ur WHERE ur.`user_id`
= '2') a WHERE r.id = a.role_id ;
```

3.3.3 explain 之 select_type

表示 SELECT 的类型，常见的取值，如下表所示:

select_type	含义
SIMPLE	简单的select查询，查询中不包含子查询或者UNION
PRIMARY	查询中若包含任何复杂的子查询，最外层查询标记为该标识
SUBQUERY	在SELECT 或 WHERE 列表中包含了子查询
DERIVED	在FROM 列表中包含的子查询，被标记为 DERIVED（衍生） MYSQL会递归执行这些子查询，把结果放在临时表中
UNION	若第二个SELECT出现在UNION之后，则标记为UNION ； 若UNION包含在FROM子句的子查询中，外层SELECT将被标记为 ： DERIVED
UNION RESULT	从UNION表获取结果的SELECT

从上往下效率越来越低

1) simple

```
explain select * from t_user;
```

2) primary subquery

```
explain select * from t_user where id = (select id from user_role where role_id
= '9 ')
```

3) derived

```
explain select a.* from (select * from t_user where id in ('1','2')) a
```

4) union

```
explain select * from t_user where id = '1' union select * from t_user where id
= '2'
```

3.3.4 explain 之 table

展示这一行的数据是关于哪一张表的

3.3.5 explain 之 type

type 显示的是访问类型，是较为重要的一个指标，可取值为:

type	含义
NULL	MySQL不访问任何表，索引，直接返回结果
system	表只有一行记录(等于系统表)，这是const类型的特例，一般不会出现
const	表示通过索引一次就找到了，const 用于比较primary key 或者 unique 索引。因为只匹配一行数据，所以很快。如将主键置于where列表中，MySQL 就能将该查询转换为一个常亮。const于将 "主键" 或 "唯一" 索引的所有部分与常量值进行比较
eq_ref	类似ref，区别在于使用的是唯一索引，使用主键的关联查询，关联查询出的记录只有一条。常见于主键或唯一索引扫描
ref	非唯一性索引扫描，返回匹配某个单独值的所有行。本质上也是一种索引访问，返回所有匹配某个单独值的所有行（多个）
range	只检索给定返回的行，使用一个索引来选择行。 where 之后出现 between ， < , > , in 等操作。
index	index 与 ALL的区别为 index 类型只是遍历了索引树， 通常比ALL 快， ALL 是遍历数据文件。
all	将遍历全表以找到匹配的行

结果值从最好到最坏以此是:

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

1) null

```
explain select now();
```

2)system

```
explain select * from (select * from t_user where id = '1') a
```

3)const

```
explain select * from t_user where id = '1'
explain select * from t_user where username = 'stu1'
```

4)eq_ref

```
explain select * from t_user u, t_role r where u.id = r.id
```

5)ref

```
create index idx_user_name on t_user(name)
explain select * from t_user where name = 'a'
```

6) index

```
explain select id from t_user
```

7)all

```
explain select * from t_user
```

一般来说， 我们需要保证查询至少达到 range 级别， 最好达到ref

3.3.6 explain 之 key

```
possible_keys ： 显示可能应用在这张表的索引， 一个或多个。
key ： 实际使用的索引， 如果为NULL， 则没有使用索引。
key_len ： 表示索引中使用的字节数， 该值为索引字段最大可能长度，并非实际使用长度，在不损失精确性的前 提下， 长度越短越好 。
```

3.3.7 explain 之 rows

扫描行的数量。

3.3.8 explain 之 extra

其他的额外的执行计划信息，在该列展示 。

extra	含义
using filesort	说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取， 称为“文件排序”, 效率低。
using temporary	使用了临时表保存中间结果，MySQL在对查询结果排序时使用临时表。常见于 order by 和 group by； 效率低
using index	表示相应的select操作使用了覆盖索引， 避免访问表的数据行， 效率不错。

1) using filesort

```
explain select * from t_user order by password
```

2) using temporary

```
explain select * from t_user group by password
```

3)using index

```
explain select username from t_user order by username
```

索引的使用

4.2.1 准备环境

```
create table `tb_seller` (
    `sellerid` varchar (100),
    `name` varchar (100),
    `nickname` varchar (50),
    `password` varchar (60),
    `status` varchar (1),
    `address` varchar (100),
    `createtime` datetime,
    primary key(`sellerid`)
)engine=innodb default charset=utf8mb4;

insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('alibaba','阿里巴巴','阿里小
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00'); insert
into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('baidu','百度科技有限公司','百度小
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('huawei','华为科技有限公司','华为小
店','e10adc3949ba59abbe56e057f20f883e','0','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('qf','千锋教育科技有限公司','千锋教
育','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('itheima','好程序员','好程序
员','e10adc3949ba59abbe56e057f20f883e','0','北京市','2088-01-01 12:00:00'); insert
into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('luoji','罗技科技有限公司','罗技小
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('oppo','OPPO科技有限公司','OPPO官方旗舰
店','e10adc3949ba59abbe56e057f20f883e','0','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('ourpalm','掌趣科技股份有限公司','掌趣小
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('qiandu','千度科技','千度小
店','e10adc3949ba59abbe56e057f20f883e','2','北京市','2088-01-01 12:00:00'); insert
into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('sina','新浪科技有限公司','新浪官方旗舰
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('xiaomi','小米科技','小米官方旗舰
店','e10adc3949ba59abbe56e057f20f883e','1','西安市','2088-01-01 12:00:00');
insert into `tb_seller` (`sellerid`, `name`, `nickname`, `password`, `status`,
`address`, `createtime`) values('yijia','宜家家居','宜家家居旗舰
店','e10adc3949ba59abbe56e057f20f883e','1','北京市','2088-01-01 12:00:00');

create index idx_seller_name_sta_addr on tb_seller(name,status,address);
```

4.2.2 避免索引失效

1). 全值匹配， 对索引中所有列都指定具体值。

该情况下，索引生效，执行效率高。

```
explain select * from tb_seller where name='小米科技' and status='1' and
address='北京市';
```

2). 最左前缀法则

如果索引了多列，要遵守最左前缀法则。指的是查询从索引的最左前列开始，并且不跳过索引中的列。匹配最左前缀法则，走索引

```
explain select * from tb_seller where name='小米科技';
explain select * from tb_seller where name='小米科技' and status='1';
explain select * from tb_seller where name='小米科技' and status='1' and
address='北京市';
```

违法最左前缀法则， 索引失效:

```
explain select * from tb_seller where status='1';
explain select * from tb_seller where status='1' and address='北京市';
```

如果符合最左法则，但是出现跳跃某一列，只有最左列索引生效:

```
explain select * from tb_seller where name='小米科技' and address='北京市';
```

3). 范围查询右边的列，不能使用索引 。

```
explain select * from tb_seller where name='小米科技' and status > '1' and
address='北京市';
```

4). 不要在索引列上进行运算操作， 索引将失效。

```
explain select * from tb_seller where left(name,2)='小米';
```

5). 字符串不加单引号，造成索引失效。

```
explain select * from tb_seller where name='小米科技' and status='0';
explain select * from tb_seller where name='小米科技' and status=0;
```

6). 用or分割开的条件， 如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及的索引都不会被用到。

示例，name字段是索引列， 而nickname不是索引列，中间是or进行连接是不走索引的 :

```
explain select * from tb_seller where name='小米科技' or nickname = '小米官方旗舰
店';
```

7) 以%开头的Like模糊查询，索引失效。

如果仅仅是尾部模糊匹配，索引不会失效。如果是头部模糊匹配，索引失效。

8) is NULL， is NOT NULL 有时索引失效。

```
explain select * from tb_seller where name is not null;
```

9) in 走索引， not in 索引失效。

```
explain select * from tb_seller where name in ('千度科技','小米科技')
explain select * from tb_seller where name not in ('千度科技','小米科技')
```

10). 单列索引和复合索引。

尽量使用复合索引，而少使用单列索引 。 创建复合索引

```
create index idx_name_sta_address on tb_seller(name, status, address);
就相当于创建了三个索引 ：
name
name + status
name + status + address
```

创建单列索引

```
create index idx_seller_name on tb_seller(name);
create index idx_seller_status on tb_seller(status);
create index idx_seller_address on tb_seller(address);
```

数据库会选择一个最优的索引(辨识度最高索引)来使用，并不会使用全部索引 。

优化sql

优化insert语句

如果需要同时对一张表插入很多行数据时，应该尽量使用多个值表的insert语句，这种方式将大大的缩减客户 端与数据库之间的连接、关闭等消耗。使得效率比分开执行的单个insert语句快。
示例， 原始方式为:

```
insert into tb_test values(1,'Tom');
insert into tb_test values(2,'Cat');
insert into tb_test values(3,'Jerry');
```

优化后的方案为 :

```
insert into tb_test values(1,'Tom'),(2,'Cat'), (3,'Jerry');
```

优化order by语句

```
CREATE TABLE `emp` (
`id` int(11) NOT NULL AUTO_INCREMENT, `name` varchar(100) NOT NULL,
`age` int(3) NOT NULL,
`salary` int(11) DEFAULT NULL, PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

insert into `emp` (`id`, `name`, `age`, `salary`) values('1','Tom','25','2300');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('2','Jerry','30','3500');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('3','Luci','25','2800');
insert into `emp` (`id`, `name`, `age`, `salary`) values('4','Jay','36','3500');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('5','Tom2','21','2200');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('6','Jerry2','31','3300');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('7','Luci2','26','2700');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('8','Jay2','33','3500');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('9','Tom3','23','2400');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('10','Jerry3','32','3100');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('11','Luci3','26','2900');
insert into `emp` (`id`, `name`, `age`, `salary`)
values('12','Jay3','37','4500');
```

```
# 没有创建索引
explain select id, age from emp order by age

# 创建 age 索引
explain select id, age from emp order by age
```

优化group by 语句

```
# 没有创建索引
explain select max(age) from emp group by age

# 创建 age 索引
explain select max(age) from emp group by age
```

优化分页查询

一般分页查询时，通过创建覆盖索引能够比较好地提高性能。一个常见又非常头疼的问题就是 limit 900000,10 ， 此时需要MySQL排序前900010 记录，仅仅返回900000 - 900010 的记录，其他记录丢弃，查询排序的代价非 常大 。

```
select * from tb_user LIMIT 900000, 10
```

优化

在索引上完成排序分页操作，最后根据主键关联回原表查询所需要的其他列内容。

```
select * from tb_user t, (select id from tb_user LIMIT 900000, 10) a where t.id
= a.id
```