Explain 用法

为什么要使用 explain？
explain 可以帮助我们分析 select 语句，让我们知道查询效率低下的原因，从而改进我们查询，让查询优化器能够更好的工作。

MySQL 查询优化器是如何工作的

MySQL 查询优化器有几个目标，但是其中最主要的目标是尽可能地使用索引，并且使用最严格的索引来消除尽可能多的数据行。最终目标是提交 SELECT 语句查找数据行，而不是排除数据行。优化器试图排除数据行的原因在于它排除数据行的速度越快，那么找到与条件匹配的数据行也就越快。如果能够首先进行最严格的测试，查询就可以执行地更快。

EXPLAIN 的每个输出行提供一个表的相关信息，并且每个行包括下面的列：

项	说明
id	MySQL Query Optimizer 选定的执行计划中查询的序列号。表示查询中执行 select 子句或操作表的顺序，id 值越大优先级越高，越先被执行。id 相同，执行顺序由上至下。

select_type 查询类型	说明
SIMPLE	简单的 select 查询，不使用 union 及子查询
PRIMARY	最外层的 select 查询
UNION	UNION 中的第二个或随后的 select 查询，不依赖于外部查询的结果集
DEPENDENT UNION	UNION 中的第二个或随后的 select 查询，依赖于外部查询的结果集
UNION RESULT	UNION 查询的结果集
SUBQUERY	子查询中的第一个 select 查询，不依赖于外部查询的结果集
DEPENDENT SUBQUERY	子查询中的第一个 select 查询，依赖于外部查询的结果集
DERIVED	用于 from 子句里有子查询的情况。MySQL 会递归执行这些子查询，把结果放在临时表里。
UNCACHEABLE SUBQUERY	结果集不能被缓存的子查询，必须重新为外层查询的每一行进行评估
UNCACHEABLE UNION	UNION 中的第二个或随后的 select 查询，属于不可缓存的子查询

项	说明
table	输出行所引用的表

type 重要的项，显示连接使用的类型，按最优到最差的类型排序	说明
system	表仅有一行(=系统表)。这是 const 连接类型的一个特例。
const	const 用于用常数值比较 PRIMARY KEY 时。当查询的表仅有一行时，使用 System。
eq_ref	除 const 类型外最好的可能实现的连接类型。它用在一个索引的所有部分被连接使用并且索引是 UNIQUE 或 PRIMARY KEY，对于每个索引键，表中只有一条记录与之匹配。
ref	连接不能基于关键字选择单个行，可能查找到多个符合条件的行。叫做 ref 是因为索引要跟某个参考值相比较。这个参考值或者是一个常数，或者是来自一个表里的多表查询的结果值。
ref_or_null	如同 ref，但是 MySQL 必须在初次查找的结果里找出 null 条目，然后进行二次查找。
index_merge	说明索引合并优化被使用了。
unique_subquery	在某些 IN 查询中使用此种类型，而不是常规的 ref： `value IN (SELECT primary_key FROM single_table WHERE some_expr)`
index_subquery	在某些 IN 查询中使用此种类型，与 unique_subquery 类似，但是查询的是非唯一性索引： `value IN (SELECT key_column FROM single_table WHERE some_expr)`
range	只检索给定范围的行，使用一个索引来选择行。key 列显示使用了哪个索引。当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符，用常量比较关键字列时，可以使用 range。
index	全表扫描，只是扫描表的时候按照索引次序进行而不是行。主要优点就是避免了排序，但是开销仍然非常大。
all	最坏的情况，从头到尾全表扫描。

项	说明
possible_keys	指出 MySQL 能在该表中使用哪些索引有助于查询。如果为空，说明没有可用的索引

项	说明
key	MySQL 实际从 possible_key 选择使用的索引。如果为 NULL，则没有使用索引。很少的情况下，MYSQL 会选择优化不足的索引。这种情况下，可以在 SELECT 语句中使用 USE INDEX（indexname）来强制使用一个索引或者用 IGNORE INDEX（indexname）来强制 MYSQL 忽略索引

项	说明
key_len	使用的索引的长度。在不损失精确性的情况下，长度越短越好

项	说明
ref	显示索引的哪一列被使用了

项	说明
rows	MYSQL 认为必须检查的用来返回请求数据的行数

extra 中出现以下 2 项意味着 MYSQL 根本不能使用索引，效率会受到重大影响。应尽可能对此进行优化

extra 项	说明
Using filesort	表示 MySQL 会对结果使用一个外部索引排序，而不是从表里按索引次序读到相关内容。可能在内存或者磁盘上进行排序。MySQL 中无法利用索引完成的排序操作称为“文件排序”。
Using temporary	表示 MySQL 在对查询结果排序时使用临时表。常见于排序 order by 和分组查询 group by。

弄明白了 explain 语法返回的每一项结果，我们就能知道查询大致的运行时间了，如果查询里没有用到索引、或者需要扫描的行过多，那么可以感到明显的延迟。因此需要改变查询方式或者新建索引。

下面来举一个例子来说明下 explain 的用法。

先来一张表：

```
CREATE TABLE IF NOT EXISTS `article` (
```

```
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`author_id` int(10) unsigned NOT NULL,
`category_id` int(10) unsigned NOT NULL,
`views` int(10) unsigned NOT NULL,
`comments` int(10) unsigned NOT NULL,
`title` varbinary(255) NOT NULL,
`content` text NOT NULL,
PRIMARY KEY (`id`)
);
```

再插几条数据：

```
INSERT INTO `article`
(`author_id`, `category_id`, `views`, `comments`, `title`, `content`) VALUES
(1, 1, 1, 1, '1', '1'),
(2, 2, 2, 2, '2', '2'),
(1, 1, 3, 3, '3', '3');
```

需求：

查询 category_id 为 1 且 comments 大于 1 的情况下，views 最多的 article_id。

先查查试试看：

```
EXPLAIN
SELECT author_id
FROM `article`
WHERE category_id = 1 AND comments > 1
ORDER BY views DESC
LIMIT 1;
```

看看部分输出结果：

select_type	type	key	ref	rows	Extra
SIMPLE	ALL	NULL	NULL	5	Using where; Using filesort

很显然，type 是 ALL，即最坏的情况。Extra 里还出现了 Using filesort，也是最坏的情况。优化是必须的。

嗯，那么最简单的解决方案就是加索引了。好，我们来试一试。查询的条件里即 where 之后共使用了 category_id，comments，views 三个字段。那么来一个联合索引是最简单的了。

```
ALTER TABLE `article` ADD INDEX x ( `category_id` , `comments`, `views` );
```

结果有了一定好转，但仍然很糟糕：

type	key	Extra
range	x	Using where; Using filesort

type 变成了 range，这是可以忍受的。但是 extra 里使用 Using filesort 仍是无法接受的。
但是我们已经建立了索引，为啥没用呢？

这是因为按照 BTree 索引的工作原理，先排序 category_id，如果遇到相同的 category_id 则再排序 comments，如果遇到相同的 comments 则再排序 views。当 comments 字段在联合索引里处于中间位置时，因为 comments > 1 条件是一个范围值（所谓 range），MySQL 无法利用索引再对后面的 views 部分进行检索，即 range 类型查询字段后面的索引无效。

那么我们需要抛弃 comments，删除旧索引 DROP INDEX x ON article;

然后建立新索引：ALTER TABLE `article` ADD INDEX y (`category_id` , `views`) ;

接着再运行查询：

select_type	type	key	ref	rows	Extra
SIMPLE	ref	y	const	2	Using where

可以看到，type 变为了 ref，Extra 中的 Using filesort 也消失了，结果非常理想。

再来看一个多表查询的例子。

首先定义 2 个表 class 和 room。

```
CREATE TABLE IF NOT EXISTS `class` (
`id` int(10) unsigned NOT NULL AUTO_INCREMENT,
`card` int(10) unsigned NOT NULL,
PRIMARY KEY (`id`)
);

CREATE TABLE IF NOT EXISTS `book` (
`bookid` int(10) unsigned NOT NULL AUTO_INCREMENT,
`card` int(10) unsigned NOT NULL,
PRIMARY KEY (`bookid`)
);
```

然后再分别插入大量数据。

然后来看一个左连接查询

```
explain select * from class left join book on class.card = book.card;
```

分析结果是：

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	ALL	NULL	NULL	1741	
1	SIMPLE	book	ALL	NULL	NULL	1851	

显然第二个 ALL 是需要我们进行优化的。

建立个索引试试看。

```
ALTER TABLE `book` ADD INDEX y ( `card`);
```

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	ALL	NULL	NULL	1741	
1	SIMPLE	book	ref	y	class.card	18	Using index

可以看到第二行的 type 变为了 ref，rows 也变成了 1741*18，优化比较明显。这是由左连接特性决定的。

LEFT JOIN 条件用于确定如何从右表搜索行，左边一定都有，所以右边是我们的关键点，一定需要建立索引。

删除旧索引

```
DROP INDEX y ON book;
```

建立新索引。

```
ALTER TABLE `class` ADD INDEX x ( `card`);
```

结果

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	index	x	NULL	1741	Using index
1	SIMPLE	book	ALL	NULL	NULL	1851	

基本无变化。

然后来看一个右连接查询

```
explain select * from class right join book on class.card = book.card;
```

分析结果是：

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	book	ALL	NULL	NULL	1851	
1	SIMPLE	class	ref	x	book.card	17	Using index

优化较明显。这是因为 RIGHT JOIN 条件用于确定如何从左表搜索行，右边一定都有，所以左边是我们的关键点，一定需要建立索引。

删除旧索引
DROP INDEX x ON class;
建立新索引
ALTER TABLE `book` ADD INDEX y (`card`);

再看看结果

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	book	index	y	NULL	1851	Using index
1	SIMPLE	class	ALL	NULL	NULL	1741	

基本无优化。

最后来看看 inner join 的情况
explain select * from class inner join book on class.card = book.card;

结果：

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	ALL	NULL	NULL	1741	
1	SIMPLE	book	ref	y	class.card	18	Using index

效果很好。

删除旧索引
DROP INDEX y ON book;

mysql> explain select * from class inner join book on class.card = book.card;

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	ALL	NULL	NULL	5	
1	SIMPLE	book	ALL	NULL	NULL	5	Using where

建立新索引。
ALTER TABLE `class` ADD INDEX x (`card`);

情况如下：

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	book	ALL	NULL	NULL	1851	
1	SIMPLE	class	ref	x	book.card	17	Using index

再建立一个原来的索引
ALTER TABLE `book` ADD INDEX y (`card`);

那么我们再来统一看一看三种 join 的结果：

mysql> explain select * from class inner join book on class.card = book.card;

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	index	x	NULL	1741	Using index
1	SIMPLE	book	ref	y	class.card	18	Using index

mysql> explain select * from class left join book on class.card = book.card;

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	index	x	NULL	1741	Using index
1	SIMPLE	book	ref	y	class.card	18	Using index

mysql> explain select * from class right join book on class.card = book.card;

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	book	index	y	NULL	1851	Using index
1	SIMPLE	class	ref	x	book.card	17	Using index

综上所述，inner join 和 left join 差不多，都需要优化右表。而 right join 需要优化左表。

我们再来看看三表查询的例子

先再建立一个表 phone。

```
CREATE TABLE IF NOT EXISTS `phone` (
    `phoneid` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `card` int(10) unsigned NOT NULL,
    PRIMARY KEY (`phoneid`)
    ) engine = innodb;
```

添加一个新索引。

```
ALTER TABLE `phone` ADD INDEX z ( `card`);
```

mysql> explain select * from class left join book on class.card=book.card left j
oin phone on book.card = phone.card;

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	class	index	x	NULL	1741	Using index
1	SIMPLE	book	ref	y	class.card	18	Using index
1	SIMPLE	phone	ref	z	book.card	9	Using index

后 2 行的 type 都是 ref 且总 rows 优化很好，效果不错。

MySql 中的 explain 语法可以帮助我们改写查询，优化表的结构和索引的设置，从而最大地

提高查询效率。当然，在大规模数据量时，索引的建立和维护的代价也是很高的，往往需要较长的时间和较大的空间，如果在不同的列组合上建立索引，空间的开销会更大。因此索引最好设置在需要经常查询的字段中。