

explain 显示了 **mysql** 如何使用索引来处理 **select** 语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。

虽然这篇文章我写的很长，但看起来真的不会困啊，真的都是干货啊！！！！

先解析一条 **sql** 语句，看出现什么内容

EXPLAIN SELECT

s.uid,s.username,s.name,f.email,f.mobile,f.phone,f.postalcode,f.address

FROM uhome_space AS s,uhome_spacefield AS f

WHERE 1

AND s.groupid=0

AND s.uid=f.uid

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	s	ALL	NULL	NULL	NULL	NULL	17022	Using where
1	SIMPLE	f	ALL	NULL	NULL	NULL	NULL	17039	Using where; Using join buffer

1. id

SELECT 识别符。这是 **SELECT** 查询序列号。这个不重要,查询序号即为 **sql** 语句执行的顺序，看下面这条 **sql**

EXPLAIN SELECT *FROM (SELECT* FROM uhome_space LIMIT 10)AS s

它的执行结果为

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	10	
2	DERIVED	uhome_space	ALL	NULL	NULL	NULL	NULL	17022	Using index

可以看到这时的 **id** 变化了

2.select_type

select 类型，它有以下几种值

2.1 simple 它表示简单的 select,没有 union 和子查询

2.2 primary 最外面的 select,在有子查询的语句中,最外面的 select 查询就是 primary,上图中就是这样

2.3 union union 语句的第二个或者说是后面那一个.现执行一条语句, explain

```
select * from uhome_space limit 10 union select * from uhome_space limit 10,10
```

会有如下结果

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	uhome_space	ALL	NULL	NULL	NULL	NULL	17022	
2	UNION	uhome_space	ALL	NULL	NULL	NULL	NULL	17022	
NULL	UNION RESULT	<union1,2>	ALL	NULL	NULL	NULL	NULL	NULL	

第二条语句使用了 union

2.4 dependent union UNION 中的第二个或后面的 SELECT 语句,取决于外面的查询

2.5 union result UNION 的结果,如上面所示

还有几个参数,这里就不说了,不重要

3 table

输出的行所用的表,这个参数显而易见,容易理解

4 type

连接类型。有多个参数,先从最佳类型到最差类型介绍 **重要且困难**

4.1 system

表仅有一行,这是 const 类型的特列,平时不会出现,这个也可以忽略不计

4.2 const

表最多有一个匹配行, 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

记住一定是用到 primary key 或者 unique,并且只检索出两条数据的情况下才会是 const,看下面这条语句

```
explain SELECT * FROM `asj_admin_log` limit 1,结果是
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	asj_admin_log	ALL	NULL	NULL	NULL	NULL	72304	

虽然只搜索一条数据，但是因为没有用到指定的索引，所以不会使用 `const`。继续看下面这个

`explain SELECT * FROM `asj_admin_log` where log_id = 111`

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	asj_admin_log	const	PRIMARY	PRIMARY	4	const	1	

`log_id` 是主键，所以使用了 `const`。所以说可以理解为 `const` 是最优化的

4.3 eq_ref

对于 `eq_ref` 的解释，mysql 手册是这样说的："对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了 `const` 类型。它用在索引的所有部分被联接使用并且索引是 `UNIQUE` 或 `PRIMARY KEY`"。`eq_ref` 可以用于使用=比较带索引的列。看下面的语句

`explain select * from uhome_spacefield,uhome_space where uhome_spacefield.uid = uhome_space.uid`

得到的结果是下图所示。很明显，mysql 使用 `eq_ref` 联接来处理 `uhome_space` 表。

id	select_type	table	type	possible_keys	key	key_len	ref	rows
1	SIMPLE	uhome_spacefield	ALL	PRIMARY	(NULL)	(NULL)	(NULL)	1703
1	SIMPLE	uhome_space	eq_ref	PRIMARY	PRIMARY	4	uspace_uhome.uhome_spacefield.uid	

目前的疑问：

4.3.1 为什么是只有 `uhome_space` 一个表用到了 `eq_ref`，并且 `sql` 语句如果变成

`explain select * from uhome_space,uhome_spacefield where uhome_space.uid = uhome_spacefield.uid`

结果还是一样，需要说明的是 `uid` 在这两个表中都是 `primary`

4.4 ref 对于每个来自于前面的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀，或如果键不是 `UNIQUE` 或 `PRIMARY KEY`（换句话说，如果联接不能基于关键字选择单个行的话），则使用 `ref`。如果使用的键仅仅匹配少量行，该联接类型是不错的。

看下面这条语句 `explain select * from uhome_space where uhome_space.friendnum = 0`，得到结果如下，这条语句能搜出 1w 条数据

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
	1	SIMPLE	uhome_space	ref	friendnum	friendnum	4	const	17006	

4.5 ref_or_null 该联接类型如同 **ref**，但是添加了 MySQL 可以专门搜索包含 NULL 值的行。在解决子查询中经常使用该联接类型的优化。

上面这五种情况都是很理想的索引使用情况

4.6 index_merge 该联接类型表示使用了索引合并优化方法。在这种情况下，**key** 列包含了使用的索引的清单，**key_len** 包含了使用的索引的最长的关键元素。

4.7 unique_subquery

4.8 index_subquery

4.9 range 给定范围内的检索，使用一个索引来检查行。看下面两条语句

```
explain select * from uhome_space where uid in (1,2)
```

```
explain select * from uhome_space where groupid in (1,2)
```

uid 有索引，**groupid** 没有索引，结果是第一条语句的联接类型是 **range**，第二个是 **ALL**。以为是一定范围所以说像 **between** 也可以这种联接，很明显

```
explain select * from uhome_space where friendnum = 17
```

这样的语句是不会使用 **range** 的，它会使用更好的联接类型就是上面介绍的 **ref**

4.10 index 该联接类型与 **ALL** 相同，除了只有索引树被扫描。这通常比 **ALL** 快，因为索引文件通常比数据文件小。（也就是说虽然 **all** 和 **Index** 都是读全表，但 **index** 是从索引中读取的，而 **all** 是从硬盘中读的）

当查询只使用作为单索引一部分的列时，MySQL 可以使用该联接类型。

4.11 ALL 对于每个来自于先前的表的行组合，进行完整的表扫描。如果表是第一个没标记 **const** 的表，这通常不好，并且通常在它情况下很差。通常可以增加更多的索引而不要使用 **ALL**，使得行能基于前面的表中的常数值或列值被检索出。

5 possible_keys 提示使用哪个索引会在该表中找到行，不太重要

6 keys MySQL 使用的索引，简单且重要

7 key_len MySQL 使用的索引长度

8 ref **ref** 列显示使用哪个列或常数与 **key** 一起从表中选择行。

9 rows 显示 MySQL 执行查询的行数，简单且重要，数值越大越不好，说明没有用好索引

10 Extra 该列包含 MySQL 解决查询的详细信息。

10.1 Distinct MySQL 发现第 1 个匹配行后，停止为当前的行组合搜索更多的行。一直没见过这个值

10.2 Not exists

10.3 range checked for each record

没有找到合适的索引

10.4 using filesort

MySQL 手册是这么解释的“MySQL 需要额外的一次传递，以找出如何按排序顺序检索行。通过根据联接类型浏览所有行并为所有匹配 **WHERE** 子句的行保存排序关键字和行的指针来完成排序。然后关键字被排序，并按排序顺序检索行。”目前不太明白

10.5 using index 只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的信息。这个比较容易理解，就是说明是否使用了索引

explain select * from ucspace_uchome where uid = 1 的 extra 为 using index (uid 建有索引)

explain select count(*) from uhome_space where groupid=1 的 extra 为 using where(groupid 未建立索引)

10.6 using temporary

为了解决查询，MySQL 需要创建一个临时表来容纳结果。典型情况如查询包含可以按不同情况列出列的 GROUP BY 和 ORDER BY 子句时。

出现 `using temporary` 就说明语句需要优化了，举个例子来说

```
EXPLAIN SELECT ads.id FROM ads, city WHERE city.city_id = 8005 AND ads.status = 'online' AND city.ads_id=ads.id ORDER BY ads.id desc
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	tbl1	index	PRIMARY	PRIMARY	1		1000	1000	
2	PRIMARY	tbl2	index	PRIMARY	PRIMARY	1		1000	1000	
3	PRIMARY	tbl3	index	PRIMARY	PRIMARY	1		1000	1000	
4	PRIMARY	tbl4	index	PRIMARY	PRIMARY	1		1000	1000	
5	PRIMARY	tbl5	index	PRIMARY	PRIMARY	1		1000	1000	
6	PRIMARY	tbl6	index	PRIMARY	PRIMARY	1		1000	1000	
7	PRIMARY	tbl7	index	PRIMARY	PRIMARY	1		1000	1000	
8	PRIMARY	tbl8	index	PRIMARY	PRIMARY	1		1000	1000	
9	PRIMARY	tbl9	index	PRIMARY	PRIMARY	1		1000	1000	
10	PRIMARY	tbl10	index	PRIMARY	PRIMARY	1		1000	1000	
11	PRIMARY	tbl11	index	PRIMARY	PRIMARY	1		1000	1000	
12	PRIMARY	tbl12	index	PRIMARY	PRIMARY	1		1000	1000	
13	PRIMARY	tbl13	index	PRIMARY	PRIMARY	1		1000	1000	
14	PRIMARY	tbl14	index	PRIMARY	PRIMARY	1		1000	1000	
15	PRIMARY	tbl15	index	PRIMARY	PRIMARY	1		1000	1000	
16	PRIMARY	tbl16	index	PRIMARY	PRIMARY	1		1000	1000	
17	PRIMARY	tbl17	index	PRIMARY	PRIMARY	1		1000	1000	
18	PRIMARY	tbl18	index	PRIMARY	PRIMARY	1		1000	1000	
19	PRIMARY	tbl19	index	PRIMARY	PRIMARY	1		1000	1000	
20	PRIMARY	tbl20	index	PRIMARY	PRIMARY	1		1000	1000	
21	PRIMARY	tbl21	index	PRIMARY	PRIMARY	1		1000	1000	
22	PRIMARY	tbl22	index	PRIMARY	PRIMARY	1		1000	1000	
23	PRIMARY	tbl23	index	PRIMARY	PRIMARY	1		1000	1000	
24	PRIMARY	tbl24	index	PRIMARY	PRIMARY	1		1000	1000	
25	PRIMARY	tbl25	index	PRIMARY	PRIMARY	1		1000	1000	
26	PRIMARY	tbl26	index	PRIMARY	PRIMARY	1		1000	1000	
27	PRIMARY	tbl27	index	PRIMARY	PRIMARY	1		1000	1000	
28	PRIMARY	tbl28	index	PRIMARY	PRIMARY	1		1000	1000	
29	PRIMARY	tbl29	index	PRIMARY	PRIMARY	1		1000	1000	
30	PRIMARY	tbl30	index	PRIMARY	PRIMARY	1		1000	1000	
31	PRIMARY	tbl31	index	PRIMARY	PRIMARY	1		1000	1000	
32	PRIMARY	tbl32	index	PRIMARY	PRIMARY	1		1000	1000	
33	PRIMARY	tbl33	index	PRIMARY	PRIMARY	1		1000	1000	
34	PRIMARY	tbl34	index	PRIMARY	PRIMARY	1		1000	1000	
35	PRIMARY	tbl35	index	PRIMARY	PRIMARY	1		1000	1000	
36	PRIMARY	tbl36	index	PRIMARY	PRIMARY	1		1000	1000	
37	PRIMARY	tbl37	index	PRIMARY	PRIMARY	1		1000	1000	
38	PRIMARY	tbl38	index	PRIMARY	PRIMARY	1		1000	1000	
39	PRIMARY	tbl39	index	PRIMARY	PRIMARY	1		1000	1000	
40	PRIMARY	tbl40	index	PRIMARY	PRIMARY	1		1000	1000	
41	PRIMARY	tbl41	index	PRIMARY	PRIMARY	1		1000	1000	
42	PRIMARY	tbl42	index	PRIMARY	PRIMARY	1		1000	1000	
43	PRIMARY	tbl43	index	PRIMARY	PRIMARY	1		1000	1000	
44	PRIMARY	tbl44	index	PRIMARY	PRIMARY	1		1000	1000	
45	PRIMARY	tbl45	index	PRIMARY	PRIMARY	1		1000	1000	
46	PRIMARY	tbl46	index	PRIMARY	PRIMARY	1		1000	1000	
47	PRIMARY	tbl47	index	PRIMARY	PRIMARY	1		1000	1000	
48	PRIMARY	tbl48	index	PRIMARY	PRIMARY	1		1000	1000	
49	PRIMARY	tbl49	index	PRIMARY	PRIMARY	1		1000	1000	
50	PRIMARY	tbl50	index	PRIMARY	PRIMARY	1		1000	1000	
51	PRIMARY	tbl51	index	PRIMARY	PRIMARY	1		1000	1000	
52	PRIMARY	tbl52	index	PRIMARY	PRIMARY	1		1000	1000	
53	PRIMARY	tbl53	index	PRIMARY	PRIMARY	1		1000	1000	
54	PRIMARY	tbl54	index	PRIMARY	PRIMARY	1		1000</		

```
1 SIMPLE    city ref    ads_id,city_id city_id 4    const          2838
100.00 Using temporary; Using filesort
```

```
1 SIMPLE    ads    eq_ref PRIMARY    PRIMARY 4    city.ads_id
1 100.00 Using where
```

这条语句会使用 **using temporary**,而下面这条语句则不会

```
EXPLAIN SELECT ads.id FROM ads, city WHERE city.city_id = 8005 AND ads.status
= 'online' AND city.ads_id=ads.id ORDER BY city.ads_id desc
```