概述MySQL优化器

原理和一些实践

周振兴/orczhou@淘宝 For 2013华东数据库技术大会

目录

- **原理概述**
- Explain Explain
- 更高效的SQL
- 优化器的一些常见错误
- MariaDB/5.6的改进

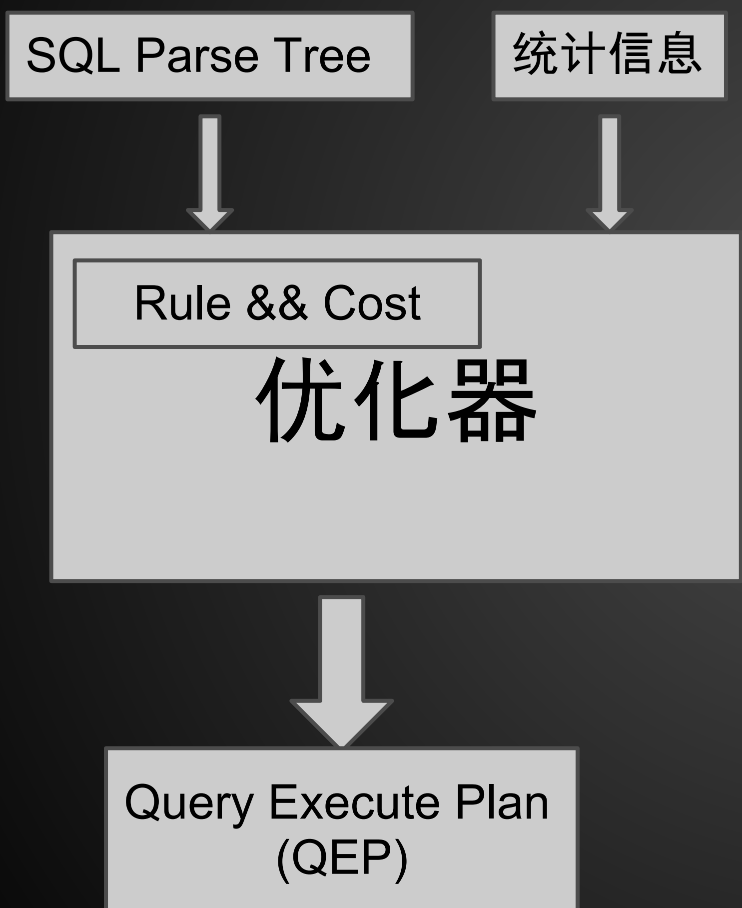
原理概述

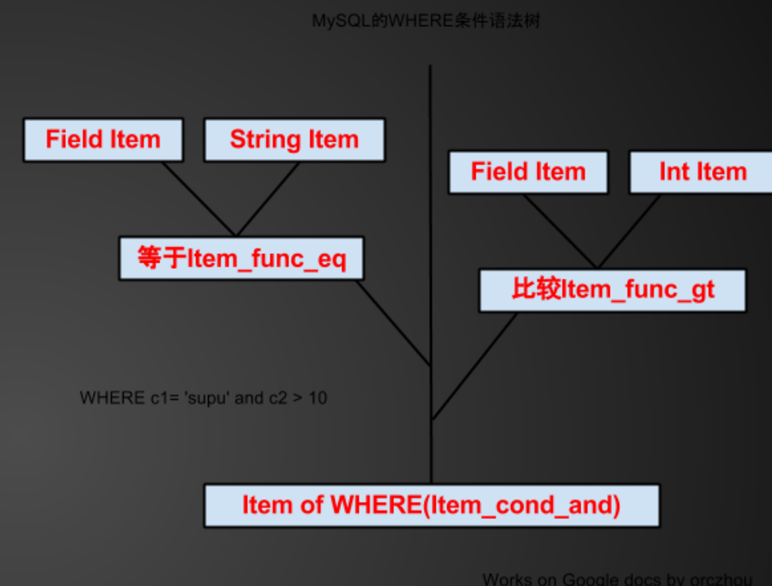
- 优化器做什么
- MySQL优化器的主要工作
- MySQL案例--优化器如何工作

术语和名词

Relational	RDBMS	通常称作
Tuple	Row	记录/行
Attribute	Column	列/字段
Restrict	Predicate	WHERE条件/限制/谓词
...	...	...
	Row-id/Rowid	记录的唯一引用/Row-id
	ROR(Rowid-ordered Retrieval)	ROR/参考

什么是优化器?





Table/Index statistics

顺序/访问方式

interface:read_first/read_next

Code show

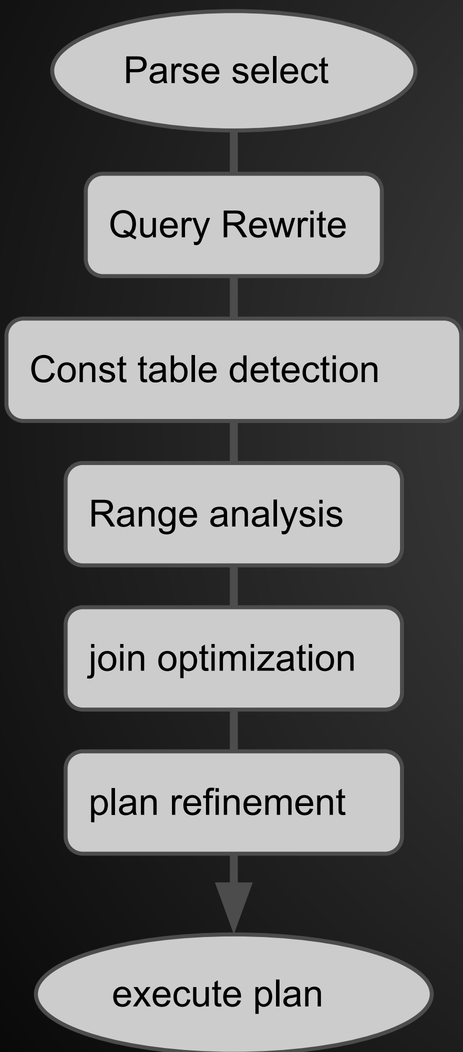
```
new JOIN(Lex);
JOIN::prepare();
JOIN::optimize();
JOIN::exec();
```

顺序和访问方式

if subquery in MySQL5.1

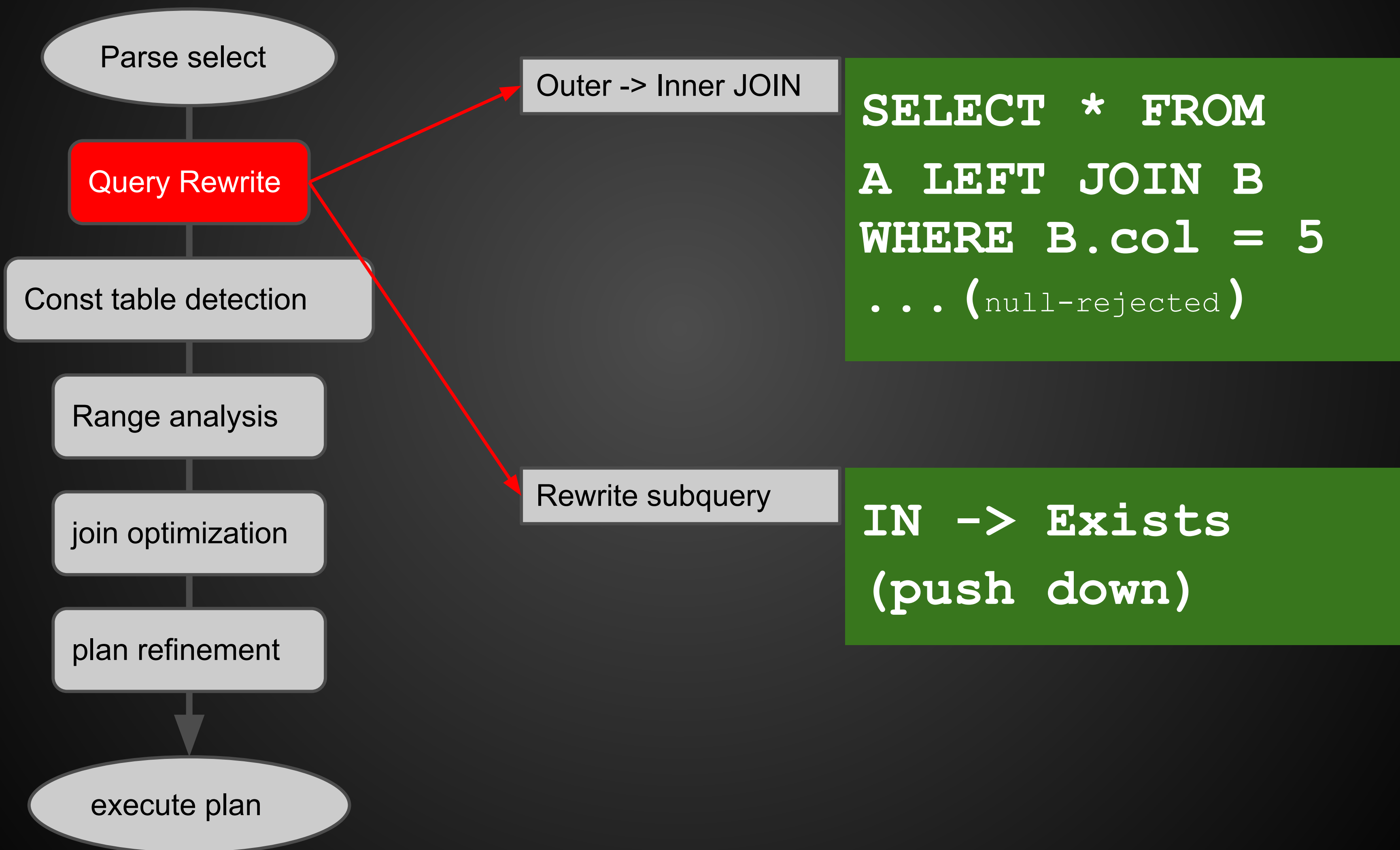
```
new JOIN(Lex);
JOIN::prepare();
JOIN::optimize();
JOIN::exec();
```

优化器的工作



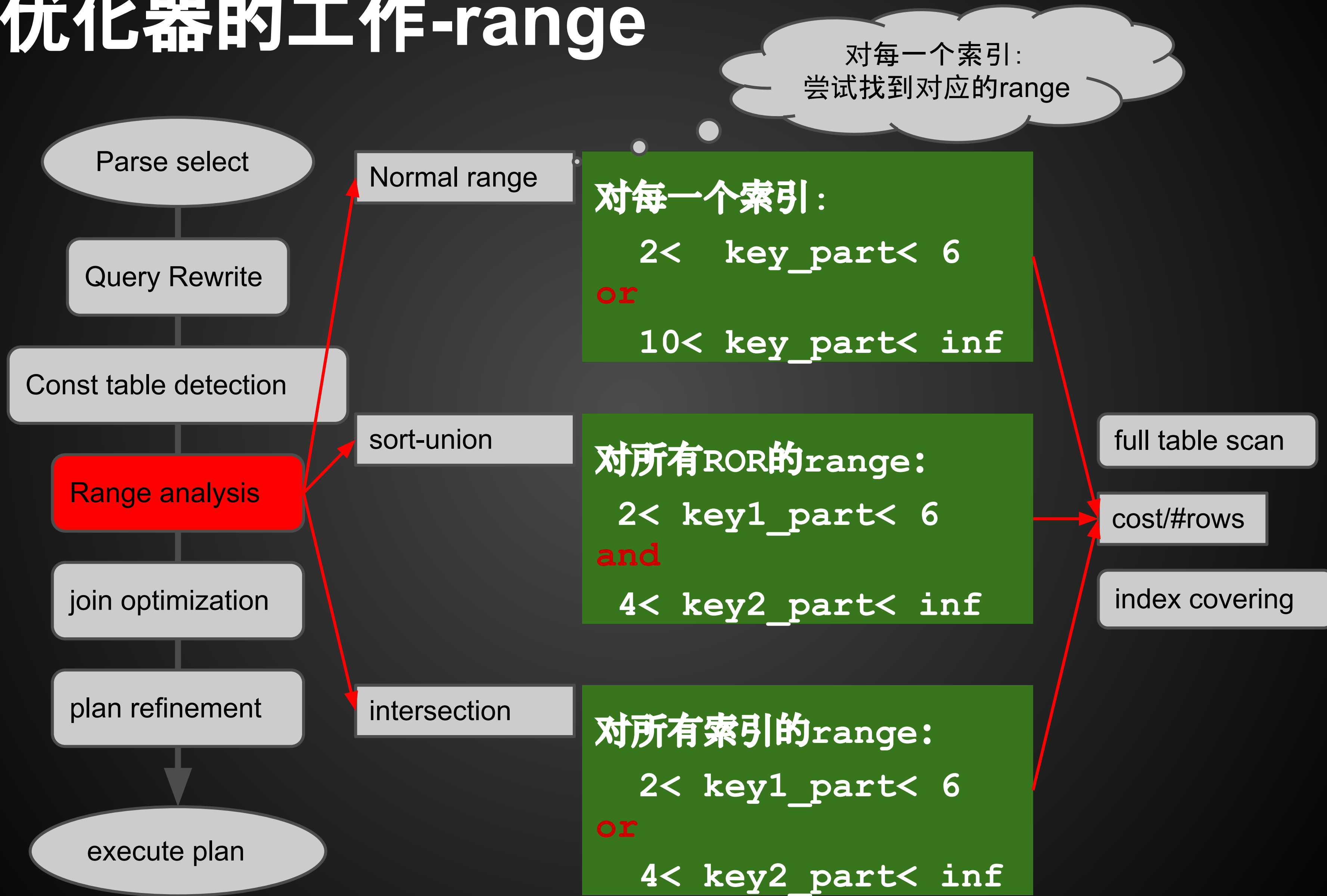
From: Sergey.P-2009-Understanding and Control of MySQL Query Optimizer

优化器的工作-Rewrite



优化器的工作-const table

优化器的工作-range



Range-analysis

cost/#rows

- Normal range

```
WHERE
   2< key1< 6
and
   4< key2< inf
```

key1: 121 / 100

key2: 178 / 147

- intersection

```
WHERE
   2< key1< 6
and
   4< key2< inf
```

key1 and key2:
(Both ROR)
254.3 / 68

- sort-union

```
WHERE
   2< key1< 6 or
   4< key2< inf
```

key1 or key2:
354.3 / 188

Cost计算

总成本 := cpu cost + io cost

MySQL如何读取/处理记录

```
info->read_record(info);           # IO COST
evaluate_join_record(join,...); # CPU COST
```

对于range类型：先根据索引找到ROWID，然后根据ROWID取出记录(一次IO)，取出后，再根据WHERE过滤(CPU消耗)

Range-analysis：normal range

- Normal range

```
WHERE
   2< key1< 6
and
   4< key2< inf
```

cost/#rows

key1: 141/100

key2: 208 / 147

table scan:150

- cost 计算sample

key1:

- storage预估返回记录数，100
- io cost = 100
- cpu cost = (#rows/5)*2 = 40

Range-analysis：normal range

- sample

```
explain select * from tmp_range
where key2_part1 > 89 and key2_part1 < 100\G
           id: 1
  select_type: SIMPLE
        table: tmp_range
         type: range
possible_keys: ind2
          key: ind2
      key_len: 4
          ref: NULL
         rows: 25
        Extra: Using where

show status like '%Last_query_cost%';
+-----------------+-----------+
| Last_query_cost | 36.009000 |
+-----------------+-----------+
```

cost: #rows + (#rows/5)*2
25 + 25/5*2 = 35

Range-analysis：intersection/交集

- Normal range

```
WHERE
  2< key1< 6
and
  4< key2< inf
```

cost/#rows

key1: 141/100

key2: 208 / 147

table scan:150

- 场景

  - 两个索引range返回rowid都很多
  - 两组rowid交集很少
  - 两个range都是ROR的(即返回的rowid都是已经排序好的)
  - 两个索引能够覆盖

Range-analysis：intersection range

- sample (参考)

```
explain select count(*) from tmp_index_merge where
(key1_part1 = 4333 and key1_part2 = 1657) and (key3_part1 = 2877)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tmp_index_merge
         type: index_merge
possible_keys: ind1,ind3
          key: ind3,ind1
      key_len: 4,8
          ref: NULL
         rows: 3622
        Extra: Using intersect(ind3,ind1); Using where; Using index
```

cost: 每个索引读取成本 + ROWID合并成本 + 合并后记录读取成本

Range-analysis：sort-union/并集

- Normal range

```
WHERE
   2< key1< 6 or
   4< key2< inf
```

cost/#rows

key1: 141/100

key2: 208 / 147

table scan:150

- 场景

  - 两个索引合并后能够覆盖整个查询

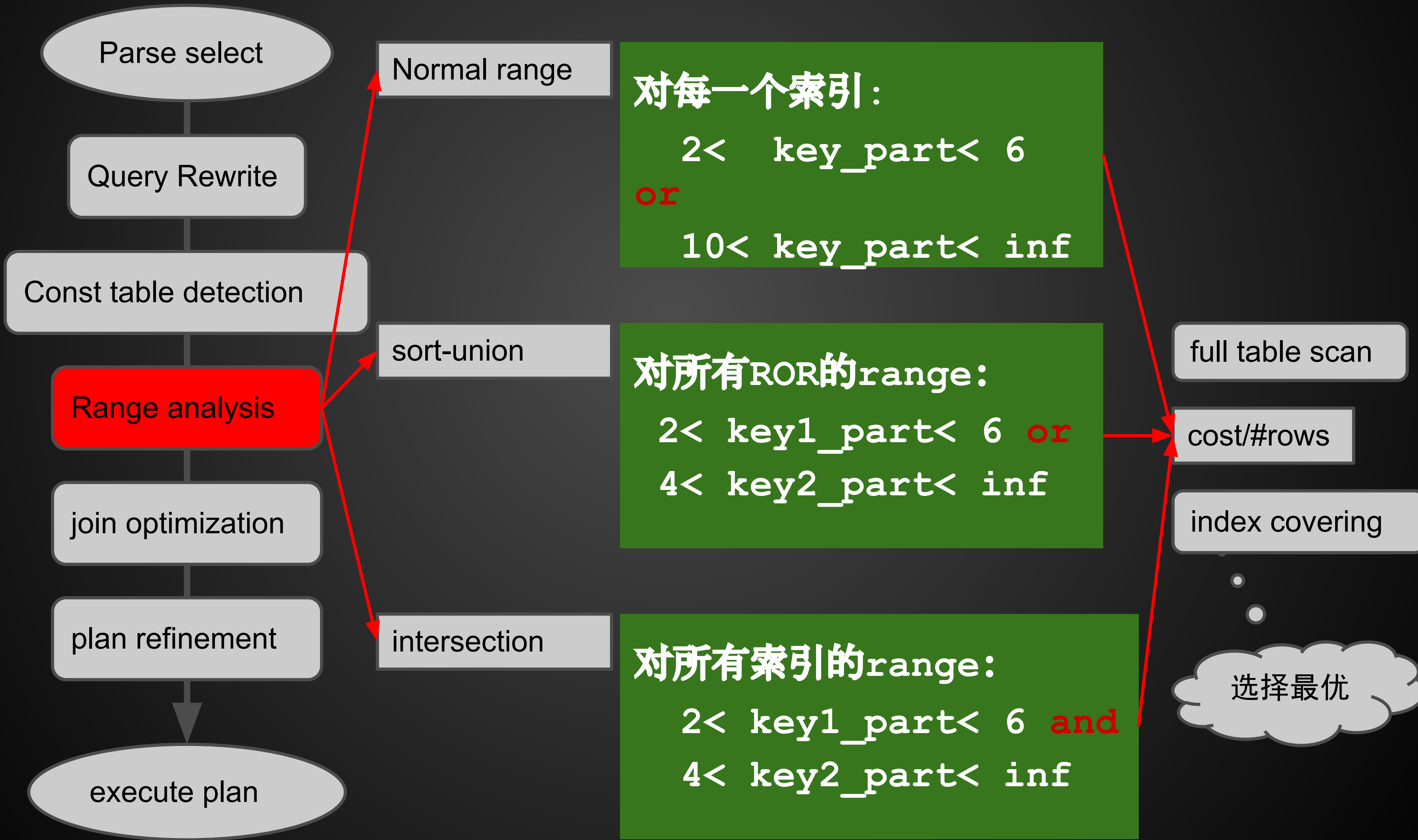
Range-analysis：sort-union

- sample (参考)

```
explain select * from tmp_index_merge where (key1_part1 = 4333 and
key1_part2 = 1657) or (key3_part1 = 2877)\G
           id: 1
  select_type: SIMPLE
        table: tmp_index_merge
         type: index_merge
possible_keys: ind1,ind3
          key: ind1,ind3
      key_len: 8,4
          ref: NULL
         rows: 2
        Extra: Using union(ind1,ind3); Using where
```

cost: 每个索引读取成本 + ROWID排序成本(非ROR)
+ 合并成本 + 合并后记录读取成本

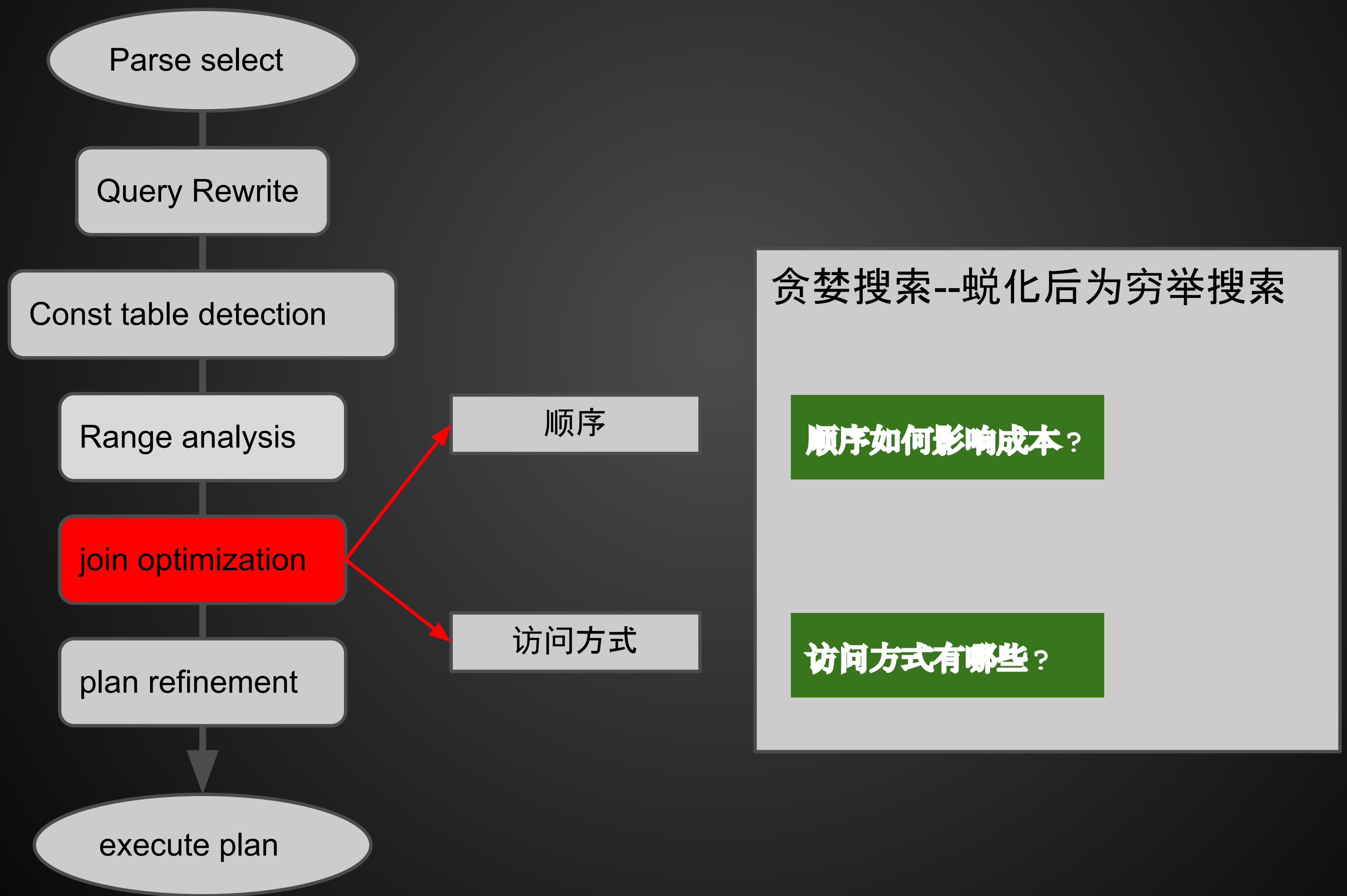
优化器的工作-range



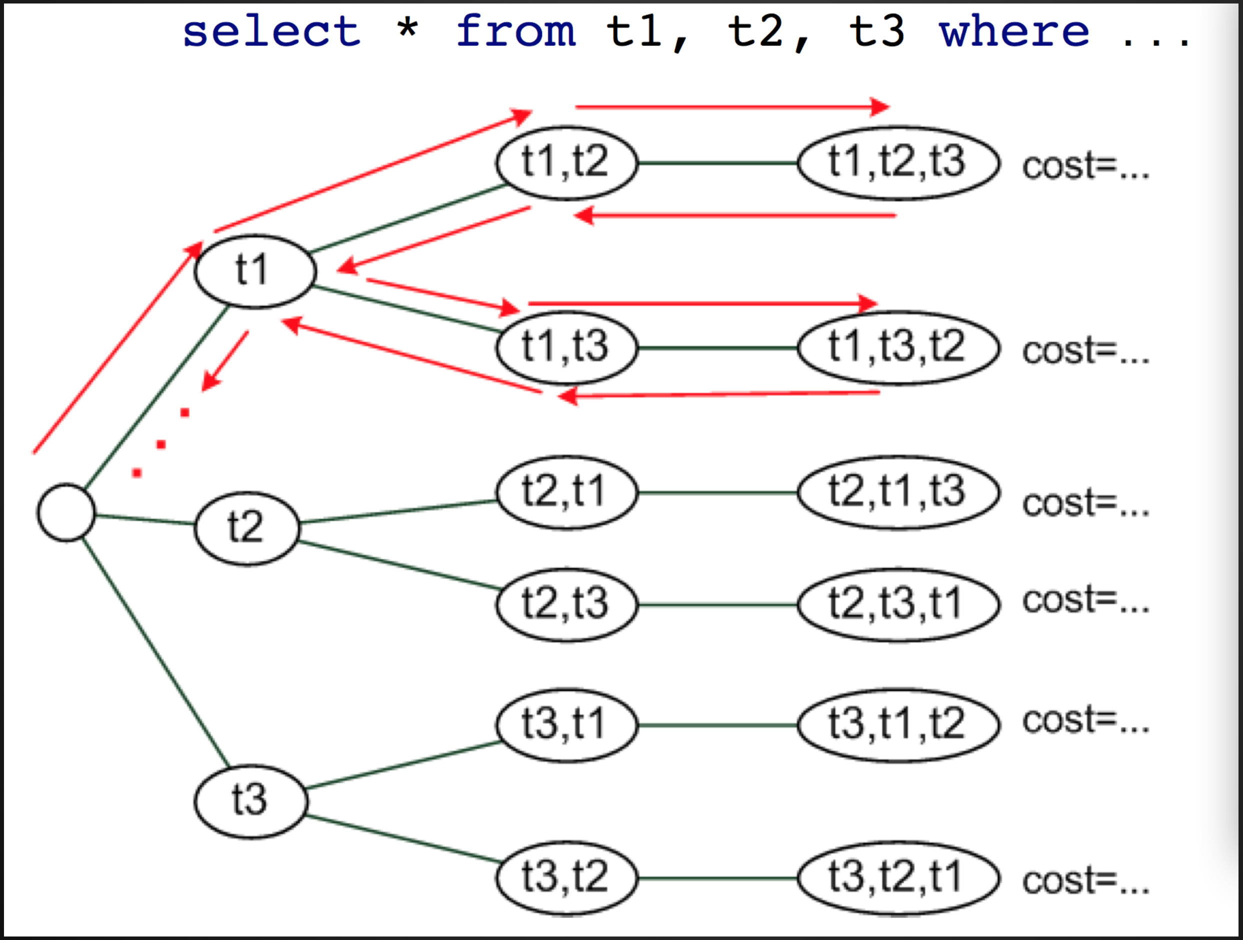
Range-analysis：其他

- range无法直接使用索引统计信息
- MySQL目前没有直方图，只能每次调用存储引擎借口(某些场景会是系统瓶颈)
- 使用存储引擎抽样借口，可以避免数据分布不均匀的问题
- 等值表达式也按照range计算，多个range一次计算

优化器的工作-join optimization



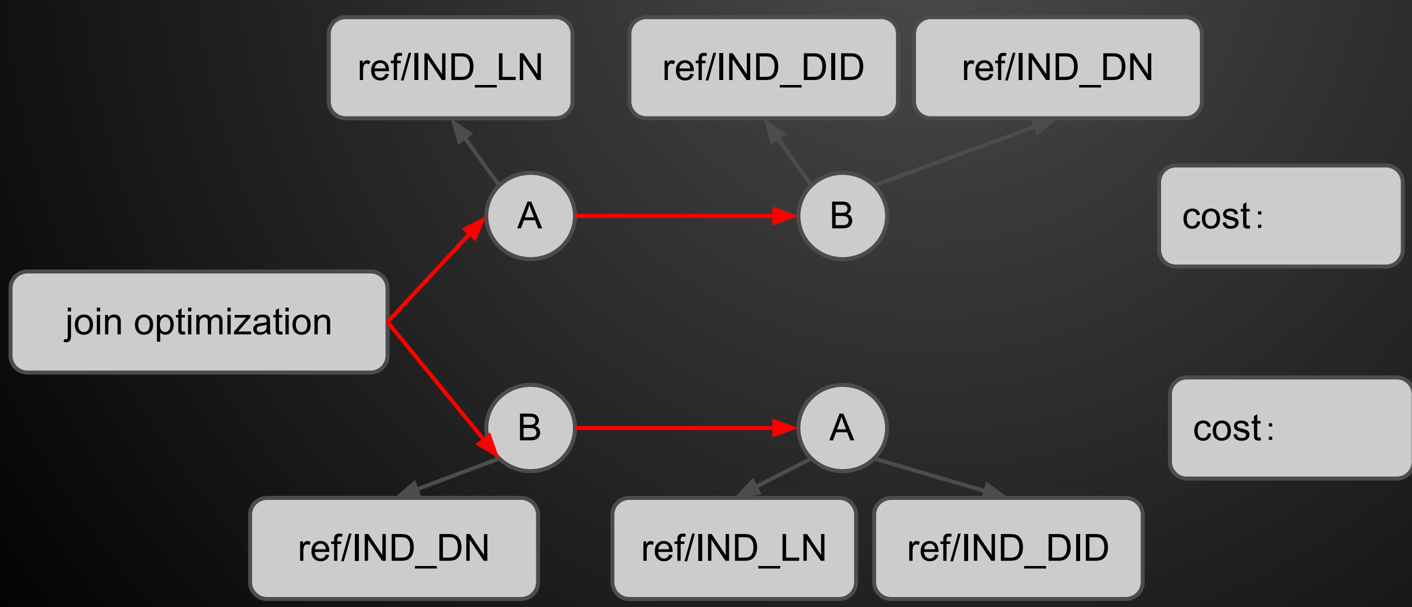
优化器的工作-join optimization



图片来源:

优化器的工作-join optimization案例

```
explain
select *
from
  employee as A,department as B
where
      A.LastName = 'zhou'
  and B.DepartmentID = A.DepartmentID
  and B.DepartmentName = 'TBX';
```



join optimization：其他

- outer join处理时，总是转换为left join
- JOIN只遍历left-deep tree
- 如果可能，outer join都转换为inner join

目录

- 原理概述
- Explain Explain
- 更高效的SQL
- 优化器的一些常见错误
- MariaDB/5.6的改进

Explain explain

- 概述/walkthrough
- 一些注意事项
  - keylen
  - select type

Explain - walkthrough

```
explain ...
********** 1. row **********
           id: 1
  select_type: SIMPLE/UNION/PRIMARY/SUBQUERY
        table: A
         type: ref/const/eq_ref/ref/...
/range/index/ALL
possible_keys: IND_L_D,IND_DID
          key: IND_L_D
      key_len: 43
          ref: const
         rows: 1
        Extra: Using where    /using index/using
```

select_type: select类型

type: 数据访问方式

rows: 预估需要扫描的记录

Extra:其他信息

Explain - key_len

```
CREATE TABLE `department` (
  `DepartmentID` int(11) DEFAULT NULL,
  `DepartmentName` varchar(20) DEFAULT NULL,
  KEY `IND_D` (`DepartmentID`),
  KEY `IND_DN` (`DepartmentName`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
********** 1. row **********
           id: 1
  select_type: SIMPLE
        table: B
         type: ref
possible_keys: IND_D,IND_DN
          key: IND_D
      key_len: 5
          ref: test.A.DepartmentID
         rows: 1
        Extra: Using where
```

key_len: 5 = INT(4 bytes) + NULL(1)

- NULL 需要额外一个字节
- VARCHAR 变成需要两个字节
- 多字节字符集:gbk*2 utf8*3
- 其他:

int	4
datetime	8
bigint	8
CHAR(M)	M*w...

参考:Data Type Storage Requirements

Explain - key_len总是取最大的

```
C
REATE TABLE `tmp_keylen` (
 `id` int(11) NOT NULL,
 `nick` char(10) DEFAULT NULL,
 `address` char(20) DEFAULT NULL,
 `color` char(10) NOT NULL,
 KEY `ind_t` (`id`,`nick`,`address`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

explain select * from tmp_keylen
where
      id >= 1
  and nick = 'zx'\G
********** 1.row **********
        table: tmp_keylen
         type: range
          key: ind_t
      key_len: 15
        Extra: Using where
```

`id >=1 and nick = 'zx'`

对MySQL来说，这是两个Range：

`id > 1`

`id = 1 and nick = 'zx'`

对应的key_len分别是 4和15

总是取最大的，所以， key_len 是15

参考：Data Type Storage Requirements

Explain - key_len看出执行计划正确性

```
explain select *
from
  tmp_users
where
      uid      = 9527
  and l_date >= '2012-12-10 10:13:17'\G
************** 1. row ******************
           id: 1
  select_type: SIMPLE
        table: tmp_users
         type: ref
possible_keys: ind_uidldate
          key: ind_uidldate
      key_len: 4
          ref: const
         rows: 418
        Extra: Using where
```

```
CREATE TABLE `tmp_users` (
  `id` int(11) NOT NULL
AUTO_INCREMENT,
  `uid` int(11) NOT NULL,
  `l_date` datetime NOT NULL,
  `data` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `ind_uidldate` (`uid`,`l_date`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

解决：使用force index

Bug#12113

Explain - type

type：JOIN过程中，单表访问方式

const	只有一条记录，如唯一索引的常数引用	WHERE primary_key=1;
ref/eq_ref/ref_or_null	引用 / 唯一索引引用 / (or null)	key = 1 / A.un_key = B.col3
range / index_merge	索引范围扫描 / 多个索引交集、并集	key > 10
index	全索引扫描	若有using index，则 索引覆盖扫描 否则，是按索引顺序扫描，再回表
ALL	全表扫描	
unique_subquery / index_subquery	IN子查询，改写成EXISTS后， 使用唯一/索引做first match扫描	
full-text	使用全文索引	
system	MyISAM表/且单表只有一条记录	

Explain - type - index_merge

使用多个索引访问数据

```
SELECT *
FROM
  tmp_index_merge
WHERE
     key1_part1 = 2
  or key2_part1 = 4
```

同时使用key1和key2获取rowid，然后merge后回表查询;一般满足:

- 两个索引访问成本都低
- 合并后成本也低于全表扫描

```
select key1_part2,key3_part1
from
  tmp_index_merge
where
     (   key1_part1 = 4333
and
         key1_part2 = 1657
     )
  and key3_part1 = 2877\G
```

同时使用key1和key2获取数据，然后merge后获得结果;需要满足:

- 所有索引访问都是ROR的
- 多个索引合并后需要是覆盖

Explain - type - index

全索引扫描

```
         type: index
possible_keys: NULL
          key: ind_uidldate
      key_len: 12
          ref: NULL
         rows: 824
        Extra: Using index
```

explain select uid from tmp_users force index (ind_uidldate)\G

如果Extra有using index，表示这是一个索引覆盖扫描，无需回表；

```
         type: index
possible_keys: NULL
          key: ind_uidldate
      key_len: 12
          ref: NULL
         rows: 824
        Extra: Using where
```

explain select data,uid from tmp_users force index (ind_uidldate) where data = 3 order by uid\G

否则，这是一个按照索引顺序的全表扫描，仍然需要回表

Explain - type - index_subquery

子查询

```
explain
select * from tmp_t1
where
  id in
     (
     select id from tmp_t2
     where age = 3
     );
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: tmp_t2
         type: index_subquery
possible_keys: ind_id,ind_age
          key: ind_id
      key_len: 5
          ref: func
         rows: 1
        Extra: Using where
```

index_subquery会使用first match原则，性能较好

子查询需要满足如下条件才会使用 index_subquery

- 子查询格式：left_exp in (Subquery)
- 优化阶段发现子查询恰好使用REF/EQ_REF
- select list对应的列也恰好是ref的字段

本案例中都满足：

- id in (...) 格式
- 子查询使用索引ind_id(id)，为ref
- select list中的列id，正是ref索引的列

unique_subquery类似于此，只是使用的唯一索引

```
CREATE TABLE `tmp_index_merge` (
  `id` int(11) NOT NULL,
  `key1_part1` int(11) NOT NULL,
  `key1_part2` int(11) NOT NULL,
  `key2_part1` int(11) NOT NULL,
  `key2_part2` int(11) NOT NULL,
  `key2_part3` int(11) NOT NULL,
  `key3_part1` int(11) NOT NULL DEFAULT '4',
  PRIMARY KEY (`id`),
  KEY `ind2` (`key2_part1`,`key2_part2`,`key2_part3`),
  KEY `ind1` (`key1_part1`,`key1_part2`,`id`),
  KEY `ind3` (`key3_part1`,`id`)
) ENGINE=InnoDB;
```

关于using index

type:index
Extra: using index; using where;

看不到的执行计划

Order by

QEP refinement

避免太多细节;