阿里数据库技术
Alibaba Database Technology

MySQL优化器的成本模型

周振兴@2016年7月

目录

成本模型与关系型数据库
简单JOIN的执行
成本计算
MySQL常见access method的成本计算
MySQL成本计算中的统计信息
成本与执行计划选择
其他的细节

成本模型与关系型数据库

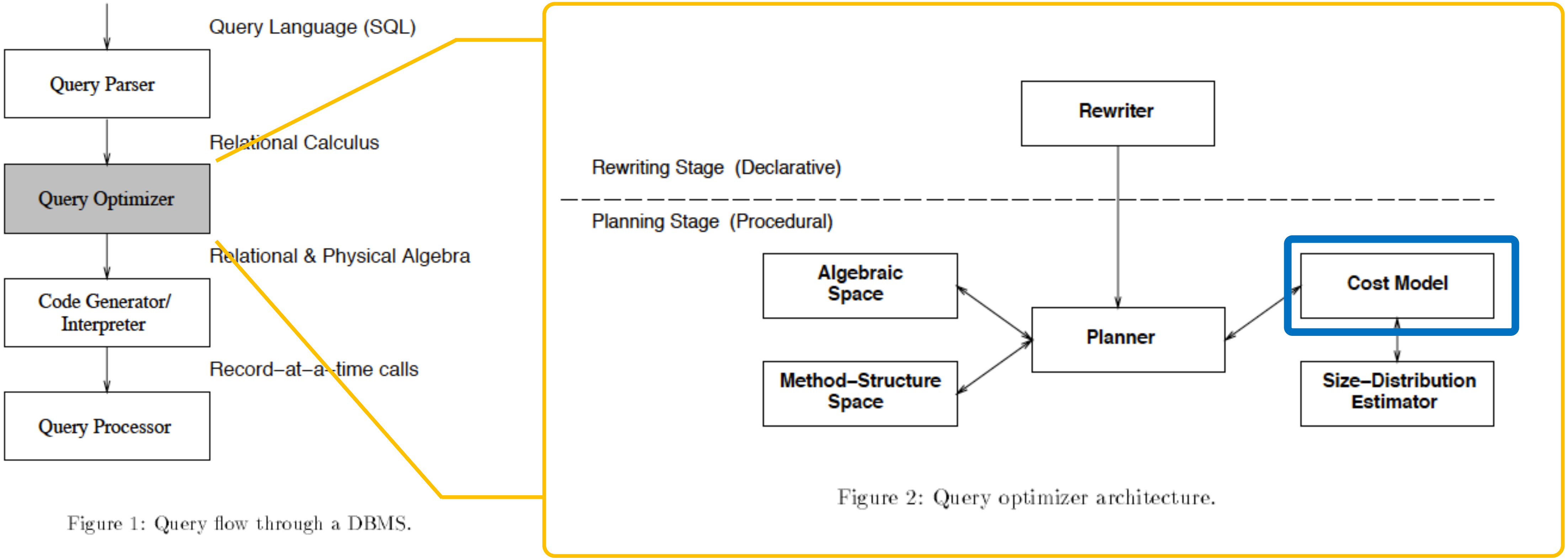


Figure 1: Query flow through a DBMS.

Figure 2: Query optimizer architecture.

图片来源： Query Optimization Yannis E. Ioannidis 1996

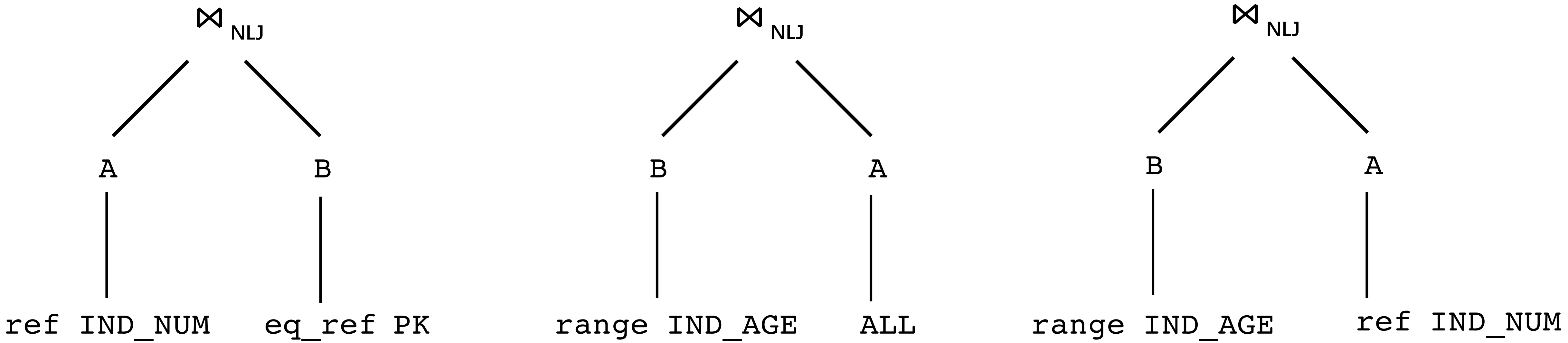
示例

```
SELECT * FROM a,b
WHERE
        a.num = 6
    and a.bid = b.id
    and b.age > 17;
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1


Table: b
CREATE TABLE `b` (
  `id` int(11) NOT NULL DEFAULT '0',
  `age` int(11) DEFAULT NULL,
  `nick` char(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_AGE` (`age`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

可能的执行计划




```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

一些事实与说明：

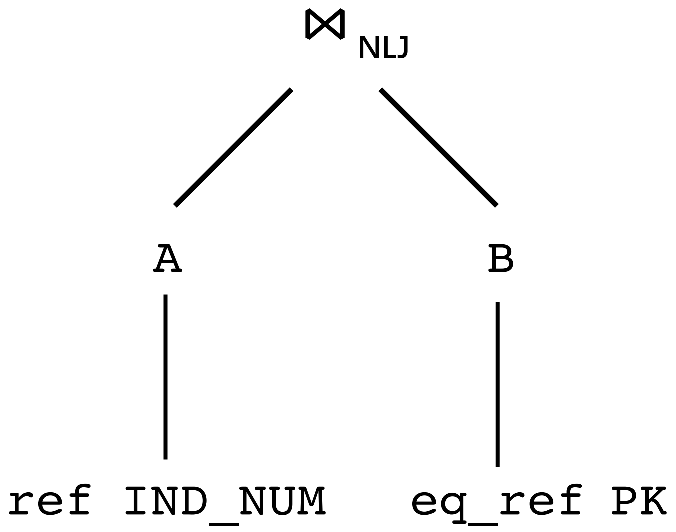
1. MySQL只支持Nested-Loop Join
2. 图示中，左边表总是Join中的outer table，右边总是inner table
   (也有说法，左边是驱动表driving table，右边是被驱动表)

简要的执行过程

for each tuple x in A with index IND_NUM

for each tuple y in B with index PK (b.id = a.bid)

if B.age > 17
return <x,y>
endif




```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

说明：tuple、row、record简单理解是指表中的一条记录

理解NLJ（一）

for each tuple x in A with index IND_NUM

1. 访问索引IND_NUM获取A.num=6的rowid
2. 根据rowid，读取A表命中的记录

for each tuple y in B with index PK (b.id = a.bid)

1. 读取主键索引中b.id = a.bid的页，取出对应tuple

if B.age > 17
return <x,y>
endif

1. 判断取出的tuple中B.age > 17

```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

理解NLJ（二）

for each tuple x in A with index IND_NUM

1. read index page (1)
2. Comparing*keys/records (131)
3. read data page (131)

for each tuple y in B with index PK (b.id = a.bid)

1. 读取主键索引页，也就是读取了数据页 (read 1 data/index page)

if B.age > 17
  return <x,y>
endif

1. evaluating query conditions

```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

目录

- 成本模型与关系型数据库
- 简单JOIN的执行
- **成本计算**
- MySQL常见access method的成本计算
- MySQL成本计算中的统计信息
- 成本与执行计划选择
- 其他的细节

成本分析

COST = COST of (**IO + CPU**)

PAGE FETCHES　　**RSI CALLS**

COST = PAGE FETCH + W * (RSI CALLS)

公式来源：Access Path Selection in a Relational Database Management System P. Griffiths Selinger... IBM 1979

MySQL成本分析

COST = COST of (**IO + CPU**)

PAGE FETCHES　　　RSI CALLS

COST = PAGE FETCH + W * (RSI CALLS)

Data Page　　Index Page　　compare key　　row evaluating　　……

MySQL成本细节

Nested Loop JOIN的成本计算

$$Cost(NLJ) = C(A) + P\_ROW(A) * C(B)$$

涉及的名词	解释
A	outer table
B	inner table
C(A)	cost of outer table
P_ROW(A)	prefix row
C(B)	cost of every time evaluating inner table

引入概念：权重W

for each tuple x in A with index IND_NUM

1. read index page (1)
2. Comparing*keys/records (131)
3. read data page (131)

for each tuple y in B with index PK (b.id = a.bid)

1. 读取主键索引页，也就是读取了数据页 (read 1 data/index page)

if B.age > 17
return <x,y>
endif

1. evaluating query conditions

```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

Cost	Cost value
Reading a random page	1.0
Evaluating query condition	0.2
Comparing key/record	0.1

(MySQL 5.7)

成本计算

for each tuple x in A with index IND_NUM

1. read index page (1)
2. Comparing*keys/records (131)
3. read data page (131)

for each tuple y in B with index PK (b.id = a.bid)

1. 读取主键索引页，也就是读取了数据页 (read 1 data/index page)

if B.age > 17
return <x,y>
endif

1. evaluating query conditions

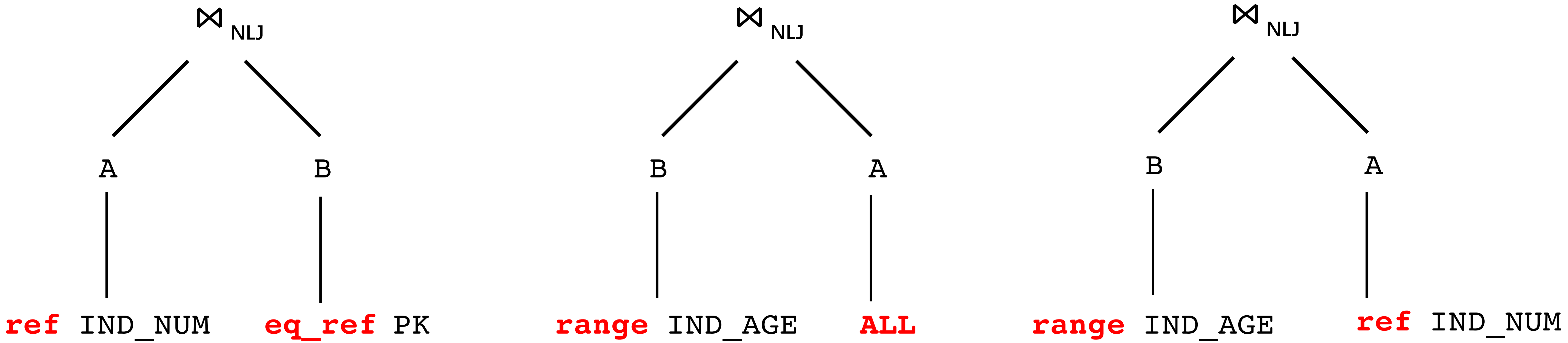
```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

*Cost(NLJ) = C(A) + P_ROW(A) * C(B)*

Cost(NLJ) = 1 + 131 + 131*0.1 + 131*（1+1*0.2）

- 成本模型与关系型数据库
- 简单JOIN的执行与成本
- 成本计算
- MySQL常见access method的成本计算
- MySQL成本计算中的统计信息
- 成本与执行计划选择
- 其他的细节

回到前面的例子




```
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

问题1：解释第二个执行计划的执行过程和成本计算？
问题2：一共有多少种执行计划？
问题3：能否列举其中的一个？

MySQL主要的access method

- table scan — where TRUE
- index scan — order by ind_a
- range scan — ind_a = 5 and ind_b > 10
- ref — where ind_a = 97 / A.ind_a = B.col
- ……

table scan的成本计算

```
SELECT * FROM a
WHERE
      a.bid < 6
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 全表扫描，逐行读取所有记录
- 评估WHERE条件是否满足

*Cost = Page(Table A) + 0.2 * ROW(Table A)*

s->table->file->scan_time()

s->table->file->stats.records;

index scan的成本计算

```
SELECT * FROM a
ORDER BY
    num
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 全索引扫描，并返回对应的rowid
- 根据rowid读取每一个记录

Cost = Page(INDEX IND_NUM) + ROW(Table A)

handler::index_only_read_time　　s->table->file->stats.records;

stats.block_size　key_length/ref_length　records

问题：如何计算索引页数

上一页问题的MySQL实现

Cost = Page(INDEX IND_NUM) + ROW(Table A)

handler::index_only_read_time　　s->table->file->stats.records;

stats.block_size　key_length/ref_length　records　　问题：如何计算索引页数

```
  @return
    Estimated cost of 'index only' scan
*/

double handler::index_only_read_time(uint keynr, double records)
{
  double read_time;
  uint keys_per_block= (stats.block_size/2/
                        (table_share->key_info[keynr].key_length + ref_length) +
                        1);
  read_time=((double) (records + keys_per_block-1) /
             (double) keys_per_block);
  return read_time;
}
```

index scan的成本计算(覆盖扫描)

```
SELECT num FROM a
ORDER BY
    num
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 全索引扫描

Cost = Page(INDEX IND_NUM)

handler::index_only_read_time

range scan的成本计算

```
SELECT * FROM a
WHERE
      num > 6 and num <10
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 读取索引范围，并返回对应的rowid
- 根据rowid读取每一个记录

*Cost = E_ROW(A) + E_ROW(A) * 0.1*

records_in_range(keynr, * min_key, * max_key)

ref的成本计算(1)

```
SELECT * FROM a
WHERE
    num = 6

(注：有索引、有取值)
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 读取索引范围，并返回对应的rowid
- 根据rowid读取每一个记录

*Cost = E_ROW(A) + E_ROW(A) * 0.1*

records_in_range(keynr, * min_key, * max_key)

ref的成本计算(2)

```
SELECT *
FROM
        b STRAIGHT_JOIN a
WHERE
      a.num = b.age and
      b.age > 10
(注：有索引、无取值)
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- 读取索引范围，并返回对应的rowid
- 根据rowid读取每一个记录

*Cost(A) = E_ROW(A) + E_ROW(A) * 0.1*

(records= keyinfo->rec_per_key[actual_key_parts(keyinfo)-1])

部分MySQL统计信息

s->table->file->scan_time()	全表扫描页数
s->table->file->stats.records	表总记录数
stats.block_size	块大小
key_length/ref_length	索引信息
records_in_range	范围中的记录数
keyinfo->rec_per_key	单个索引值引用的rowid数量
…	…

更新策略：

- ANALYZE TABLE
- SHOW TABLE STATUS
- 第一次访问表
- 访问表：
  - INFORMATION_SCHEMA.TABLES
  - INFORMATION_SCHEMA.STATISTICS
- 在变更记录数超过1/16的时候

更新策略的控制：

- innodb_stats_on_metadata

小节

- 至此，我们知道了：
  - 各种单表各种access method的成本计算方法
  - 两个表做NJL的成本计算方法

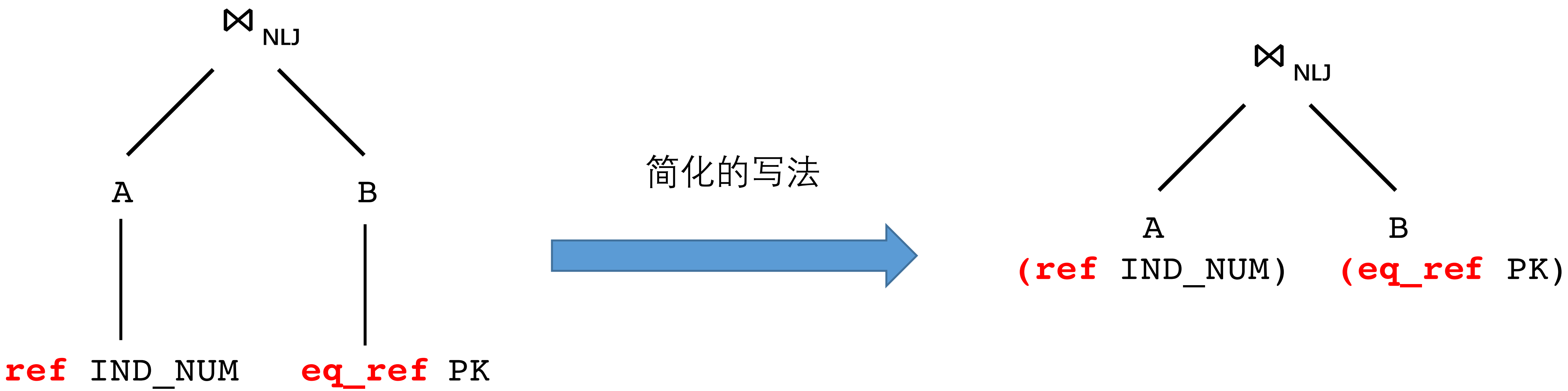
- 那么，进一步，我们可以计算：
  - 多表NLJ的成本计算：这个是一个递归计算
  - 我们可以比较不同的执行计划的成本差异

目录

- 成本模型与关系型数据库
- 简单JOIN的执行
- 成本计算
- MySQL常见access method的成本计算
- MySQL成本计算中的统计信息
- **成本与执行计划选择**
- 其他的细节

三个表JOIN的场景 → N个表JOIN的场景

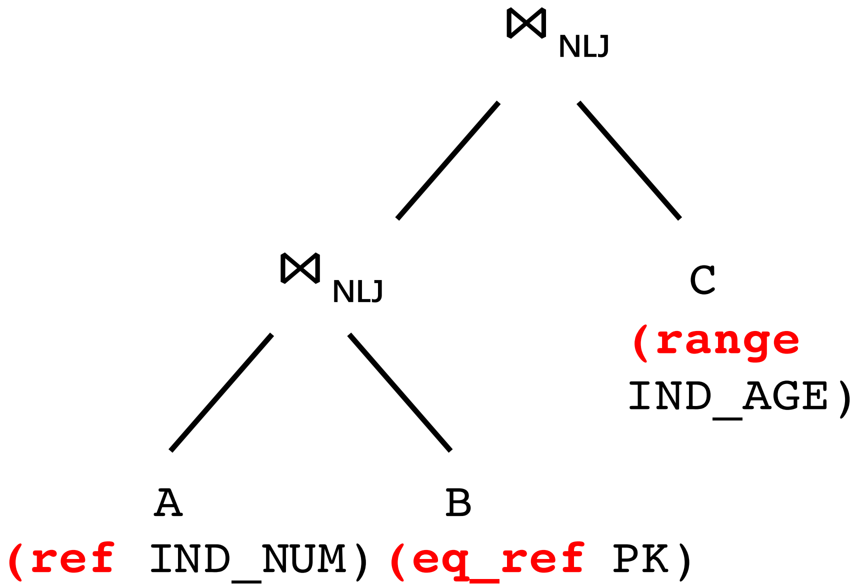
约定：简化的写法

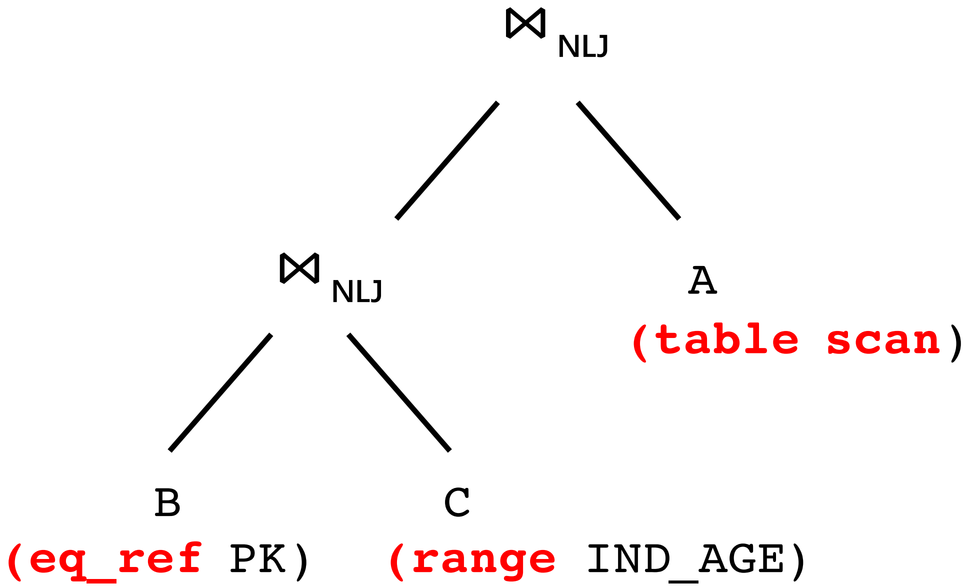


示例

```
SELECT * FROM A,B,C
WHERE
          A.num = 6
      and B.id  = 100
      and C.age > 17
      and A.cid = C.id
      and B.aid = A.id
```

可能的执行计划





一共有多少个这样的执行计划？

阿里数据库技术
Alibaba Database Technology

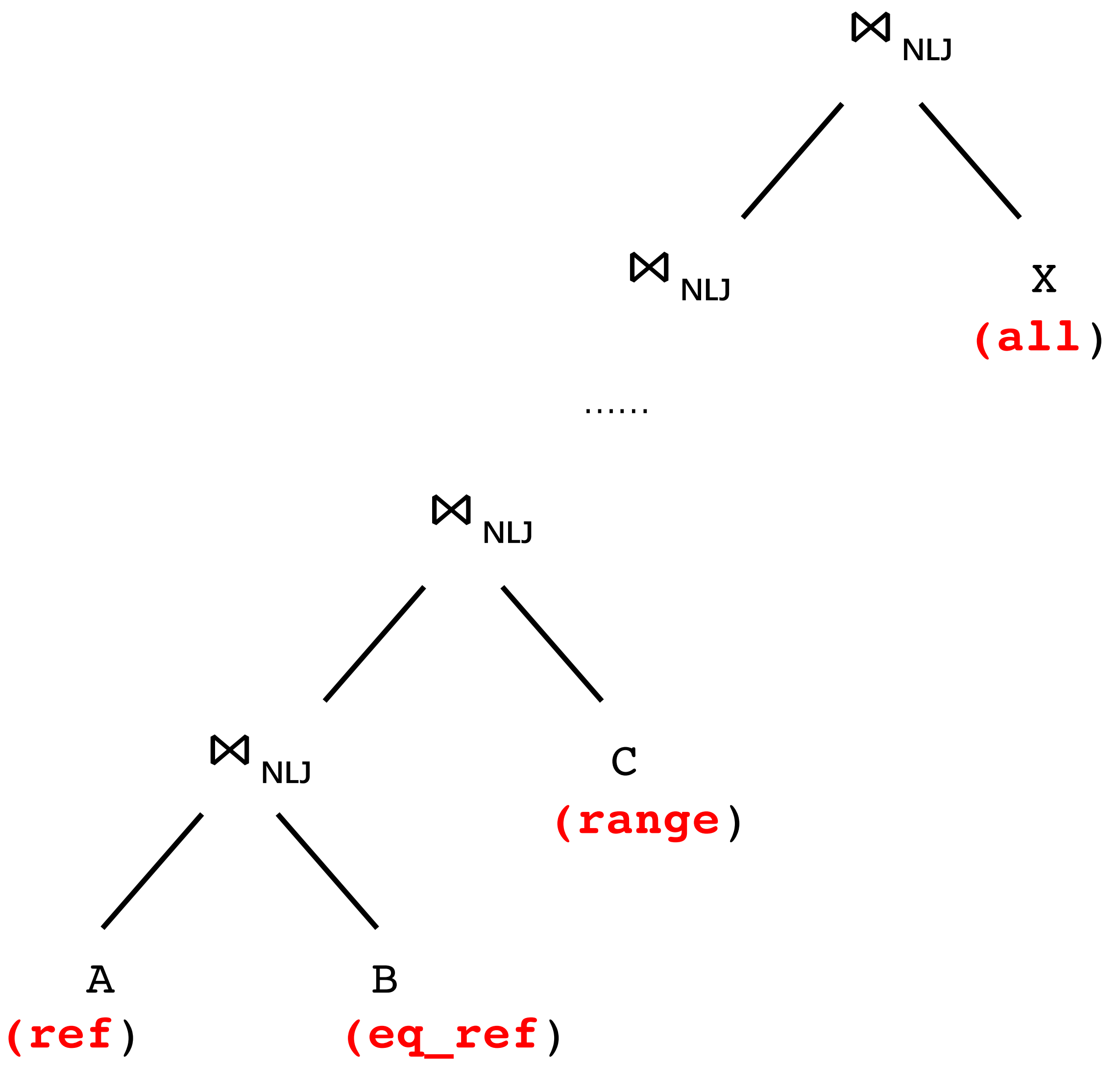
N个表的执行计划

如何找到这个问题的最优解？

1. 穷举　复杂度：O(N!)

2. 贪婪搜索　复杂度

3. 启发式(heuristics)的搜索



注：简化了如下场景
- 只考虑NLJ，不考虑sort-merge和hash join
- 没有加入关于interesting order的情况

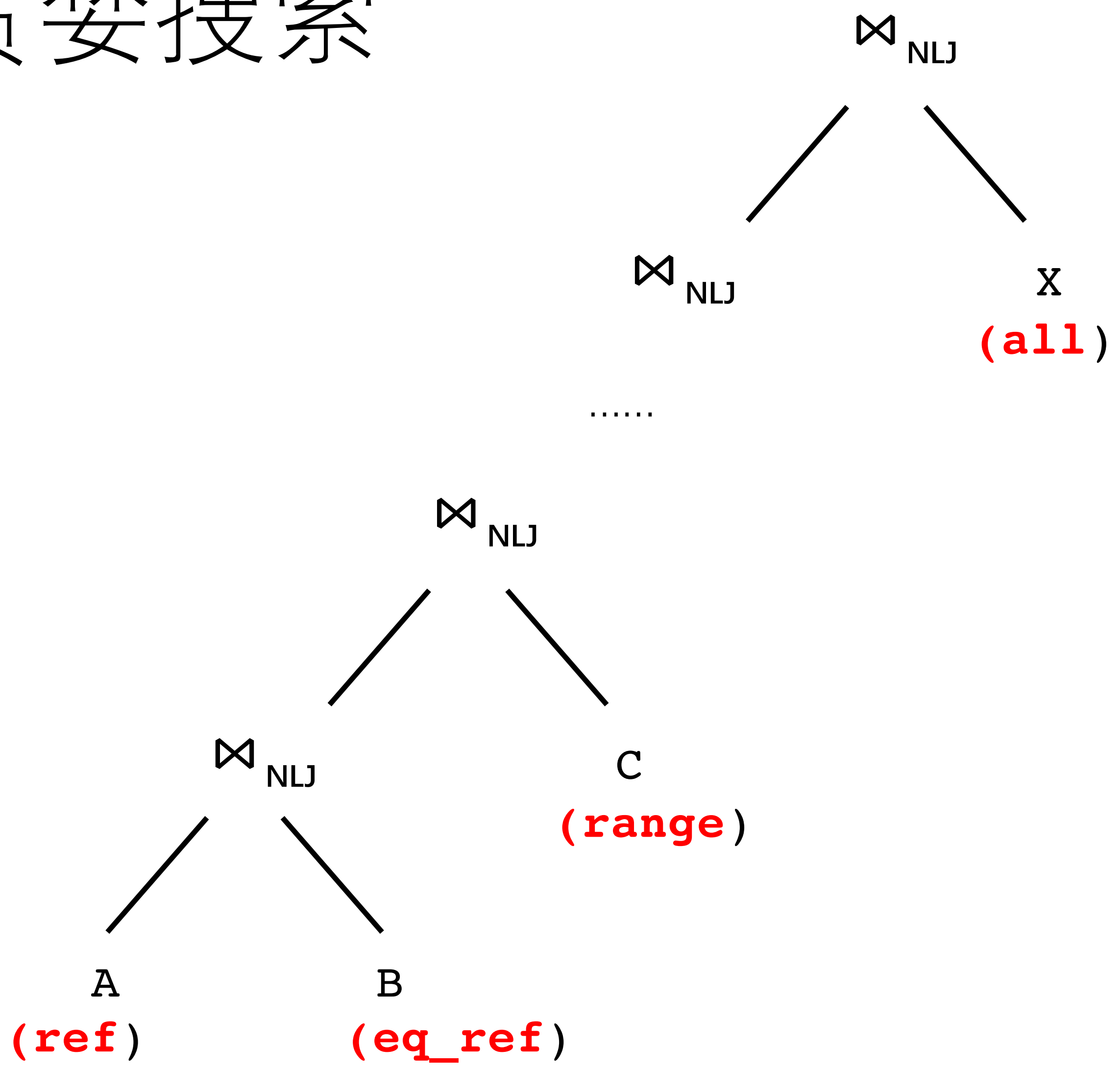
N个表的执行计划-贪婪搜索

如何找到这个问题的最优解？

1. 穷举　复杂度：O(N!)

2. 贪婪搜索 复杂度

3. 启发式(heuristics)的『裁枝』



注：简化了如下场景

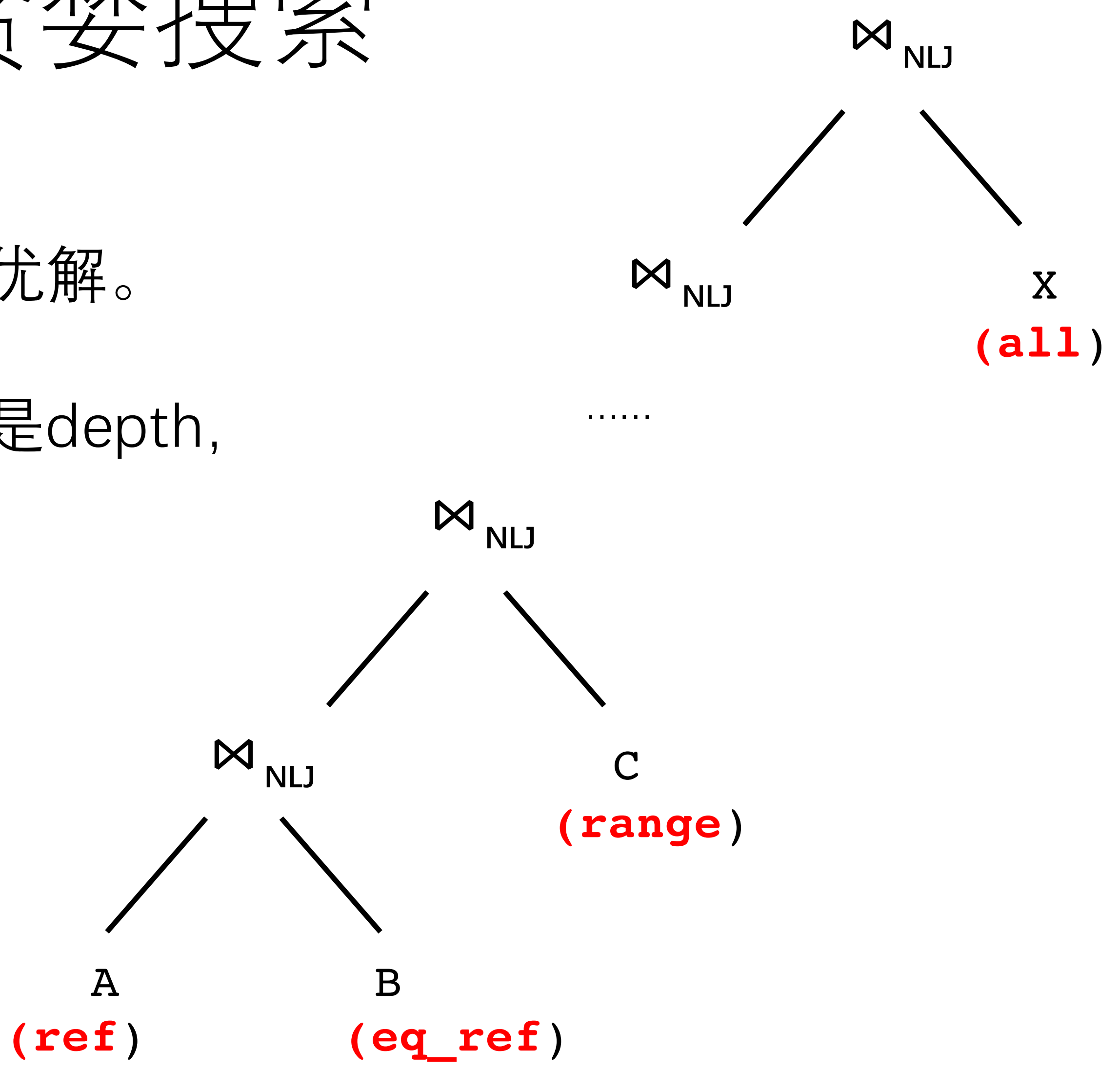
- 只考虑NLJ，不考虑sort-merge和hash join
- 没有加入关于interesting order的情况

N个表的执行计划-贪婪搜索

如何贪婪：把局部最优解当做全局最优解。

这里假设『局部最优解』的计算深度是depth，
那么复杂度为：

$$O\left(\frac{N * N^{\text{depth}}}{\text{depth}}\right)$$



问题：如果depth=1，蜕化后的情况是怎样的？

N个表的执行计划-贪婪搜索

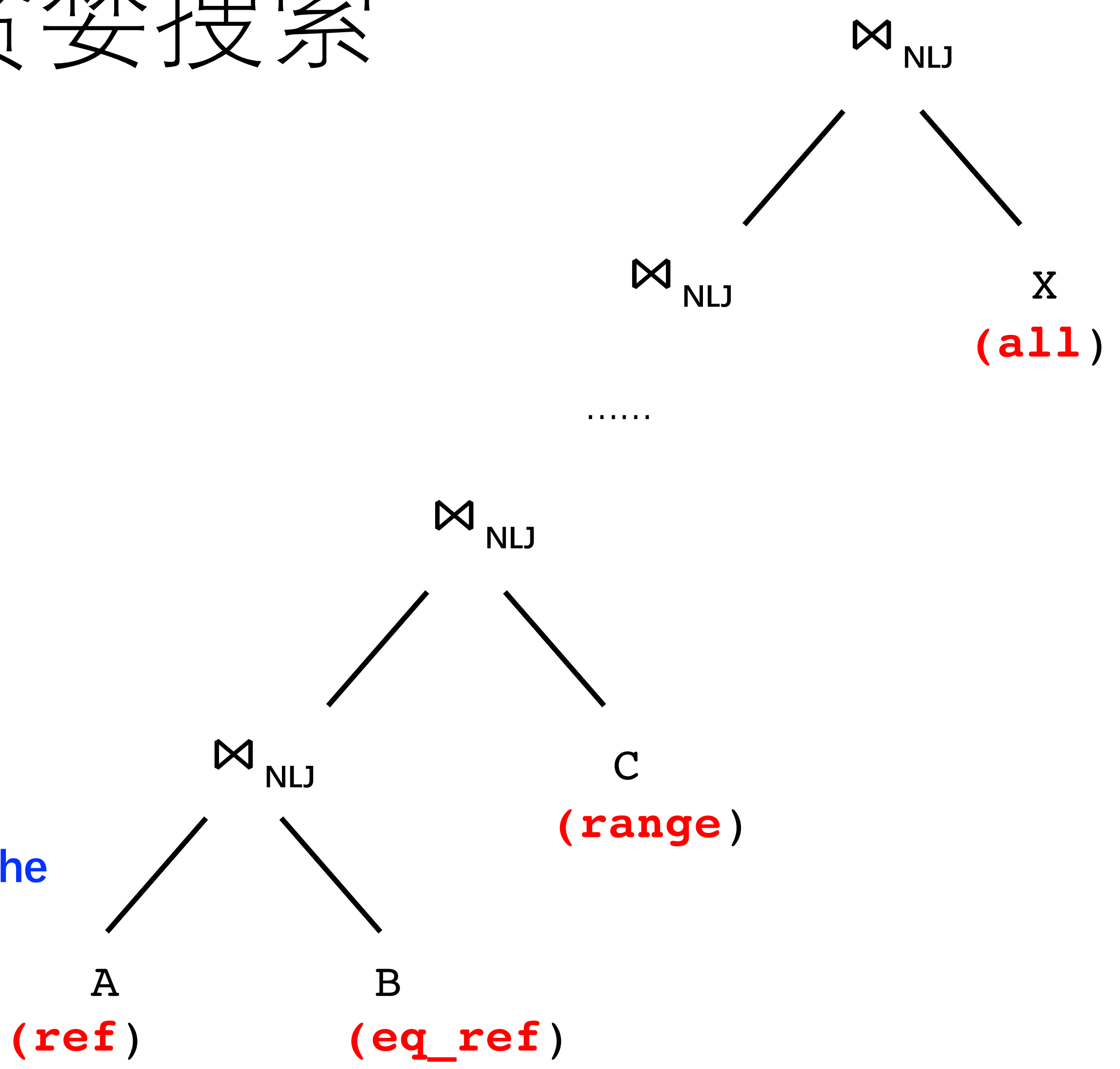
如何找到这个问题的最优解？

1. 穷举　复杂度：O(N!)

2. 贪婪搜索 复杂度

3. 启发式(heuristics)的『裁枝』

skip certain plans based on estimates of the number of rows accessed for each table

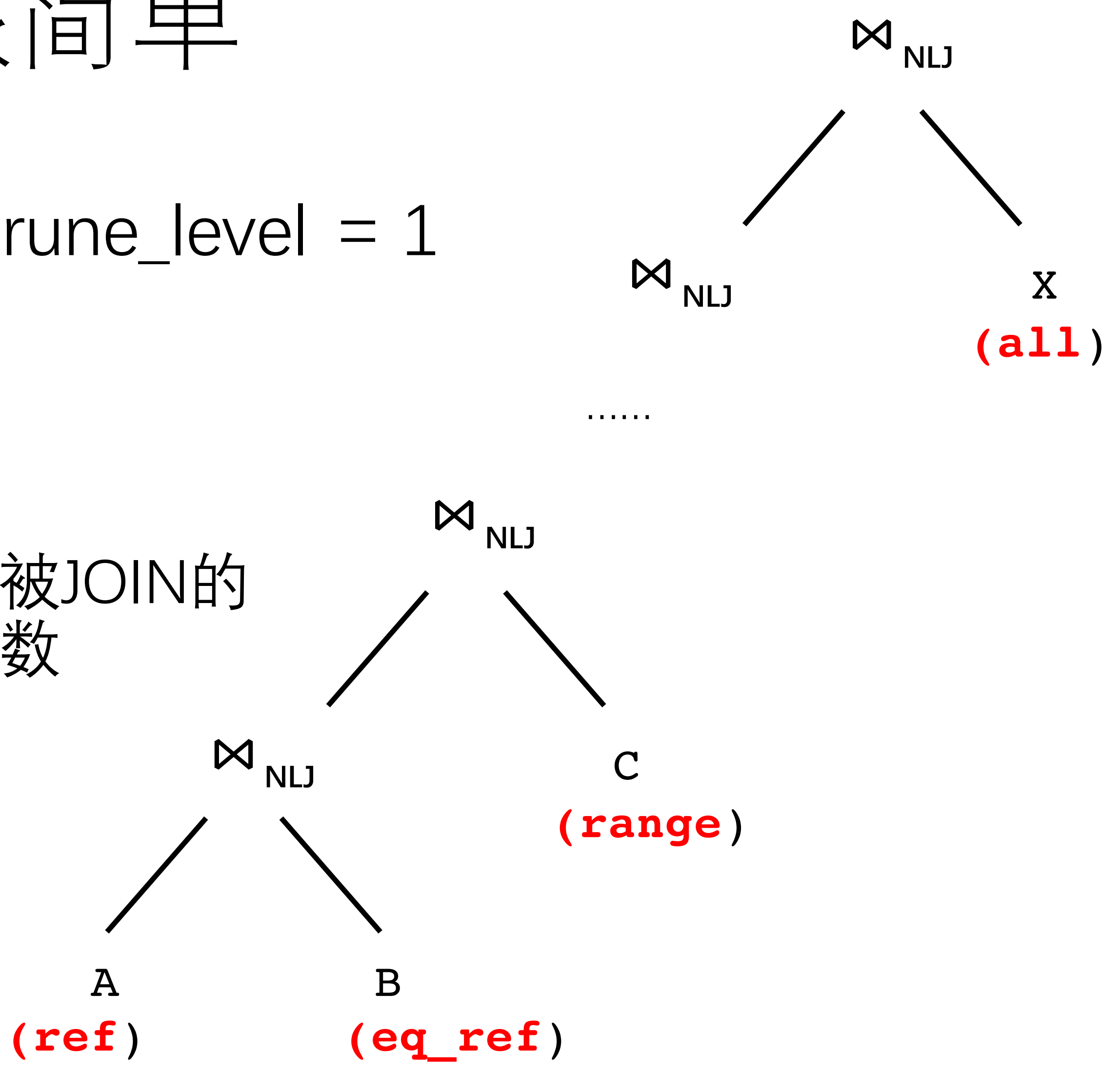


注：简化了如下场景

- 只考虑NLJ，不考虑sort-merge和hash join
- 没有加入关于interesting order的情况

理论很复杂，实际很简单

- 一般的，N < depth = 64，且prune_level = 1
  - 基本上都是穷举
  - 贪婪搜索过程中，要选择下一个被JOIN的表的时候，只看这个表返回的行数



看起来简单，但细节非常多

- 这里只考虑的NLJ，忽略sort-merge和hash join
- 没有考虑NLJ的一些优化
  - Block Nested Loops Join (MySQL)
- 为了简化，忽略了『interesting order』(order by/group by等)
- 没有讨论为什么总是left-deep tree
- 没有考虑nested query(subquery)的成本计算或者semi-join转换
- 为了简化，没有考虑多个谓词，对prefix row的影响(filter)
- 没有考虑condition_fanout_filter (MySQL5.7)
- 没有讨论GROUP BY/ORDER BY/DISTINCT等优化

参考和扩展阅读

- Paper
  - Query Optimization Yannis E. Ioannidis 文章链接
  - Access Path Selection in a Relational Database Management System P. Griffiths Selinger… IBM
- 一些slide：
  - MySQL queryoptimizer internalsand upcomingfeatures in v. 5.2
  - Implementing Joins Implementation of Database Systems
  - MySQL Cost Model
- 其他
  - MySQL Internals Manual
  - MySQL source code
  - MySQL查询优化浅析 何登成

示例

```
SELECT * FROM a,b
WHERE
        a.num = 6
    and a.bid = b.id
    and b.age > 17;
```

```
Table: a
CREATE TABLE `a` (
  `id` int(11) NOT NULL DEFAULT '0',
  `num` int(11) DEFAULT NULL,
  `bid` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_NUM` (`num`))
ENGINE=InnoDB DEFAULT CHARSET=latin1


Table: b
CREATE TABLE `b` (
  `id` int(11) NOT NULL DEFAULT '0',
  `age` int(11) DEFAULT NULL,
  `nick` char(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `IND_AGE` (`age`))
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

数据

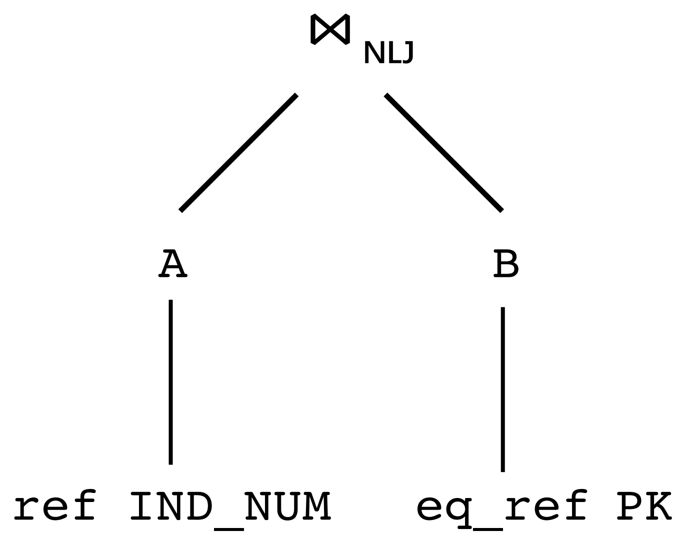
```
for i in `seq 0 5000`;do mysql -v -uroot test -e “insert into a values
(rand()*10000,rand()*10000,rand()*10000)”; done

select substring(md5(concat("adfasdfasfasdf",rand()*1000000)),1,rand()*10);

for i in `seq 0 500`;do mysql -v -uroot test -e "insert into b values
(rand()*100000,rand()*100000,substring(md5(concat('adfasdfasfasdf',rand()*100
0000)),1,rand()*10))"; done
```

附录2：Blocked Nested-Loop Join

```
for each tuple x in A with index IND_NUM
    store used columns from A in join buffer
    for each tuple y in B with index PK (b.id = a.bid)
        for each items z in join buffer
        if B.age > 17
            return <z,y>
        endif
```




```sql
SELECT * FROM A,B
WHERE
        A.num = 6
    and A.bid = B.id
    and B.age > 17;
```

inner table被扫描的次数：
(S * C)/join_buffer_size + 1

S Size of (x interesting column)
C Row return from A

说明：tuple、row、record简单理解是指表中的一条记录