MySQL Optimizer Trace

By 邱伟胜 @yhd
www.noodba.com

本ppt引用了以下文章较多的内容，在ppt中不再一一注明：

- 《 MySQL Cost Model 》by Olav Sandstå
- **《 MySQL查询优化浅析》by 何登成**
- **《 Cost Based Oracle Fundamentals 》**
- 《数据库查询优化器的艺术》

介绍内容

- 单表选择率
- 索引选择率
- RBO与CBO
- CBO
  - CBO基础
  - Cost Estimates
  - Input to Cost Model
- 统计信息
- 推荐阅读
- Optimizer Trace
  - **OPTIMIZER_TRACE的过程**
  - **join_preparation**
  - **join_optimization**
  - **代价估算实例**
  - 总结

单表选择率

在某个会议中需要召集1200名听众，问其中有多少的生日是在12月？

There are 12 possible months in the year. -- known reference

Dates of birth are (probably) evenly scattered through the year. - assumption

One-twelfth of the audience will be born in any one month. -- month’s selectivity

The request was for one specific month. - predicate

The requested month does actually exist in the calendar. -- boundary check

There are 1,200 people in the audience. -- base cardinality

The answer is one twelfth of 1,200, which is 100. -- computed cardinality

索引选择率

```
Create table t1 (a int primary key, b int, c int, d int);
create index idx_bc on t1 (b, c);
insert into t1 values
(1,1,1,1),(2,1,1,2),(3,2,1,3),(4,3,3,4),(5,3,3,5),(7,3,4,7),(8,3,5,8),(9,1,1,9);

mysql> select  table_name,index_name,stat_name,stat_value from
innodb_index_stats where table_name like 't1%';
+------------+------------+--------------+------------+
| table_name | index_name | stat_name    | stat_value |
+------------+------------+--------------+------------+
| t1         | idx_bc     | n_diff_pfx01 |          3 |
| t1         | idx_bc     | n_diff_pfx02 |          5 |
| t1         | idx_bc     | n_diff_pfx03 |          8 |
| t1         | idx_bc     | n_leaf_pages |          1 |
| t1         | idx_bc     | size         |          1 |
+------------+------------+--------------+------------+
```

selectivity (X and Y and Z) =selectivity((X and Y) and Z) =selectivity(X and Y) * selectivity(Z) =selectivity(X) * selectivity(Y) * selectivity(Z)

CBO与RBO

There are two types of optimizers:

- Cost-based
- Rule-based

Cost-based optimizers estimates the execution cost of performing a query various ways. It *tries to choose the* lowest execution cost.

Rule-based optimizers based on a set of rules. Rule-based optimizers are quicker than costbased optimizers but cost-based optimizers usually offer better performance. There are too many exceptions to the rules that the calculation overhead of cost-based optimizations is worthwhile.

CBO基础

General idea:

- Assign cost to operations
- Assign cost to partial or alternative plans
- Search for plan with lowest cost

The main cost-based optimizations:

- Fetch data method:
  - Table access
  - Index access
- Join order
- Join buffering strategy
- Subquery strategy

Cost Estimates

Cost unit:

- read of a random data page

Main cost factors:

- IO cost:
  - #pages read from table
  - #pages read from index
- CPU cost
  - Evaluating query conditions
  - Comparing keys/records;Sorting keys

Main cost constants:

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

Input to Cost Model

IO-cost:

- Estimates from storage engine based on number of pages to read
- Both index and data pages

Schema(data dictionary):

- Length of records and keys
- Uniqueness for indexed
- Nullability

Statistics:

- Number of records in table
- Key distribution/Cardinality
  - Average number of records per key value
  - Only for indexed columns
- Number of records in an index range
- Size of tables and indexes

MySQL Server层统计信息

- **CONST统计信息**
  - 此类统计信息，在表创建之后，就基本维持不变，类似于常量(非完全不变)

- 种类
  - max_data_file_length、data_file_name、**block_size... 不变**
  - **block_size**
    - 计算索引覆盖扫描Cost所需，页面大小
  - **rec_per_key... 会变化**
    - 标识一个索引键**(包括前缀键值)相同相同取值的平均个数**
    - 算法：*rec_per_key = total_rows / key_distinct_count*
    - 此参数，是MySQL进行Join Optimize的基础

- 收集策略
  - 表第一次open
  - analyze命令
  - 由InnoDB收集，并返回MySQL Server

MySQL Server层统计信息

- **VARIABLE统计信息**
  - 此类统计信息，随着记录的U/D/I操作，会发生显著的变化

- 种类
  - **records： 记录数量**
    - 直接从InnoDB的统计信息中复制，不重新收集

      ```
      n_rows = ib_table->stat_n_rows;
      stats.records = (ha_rows)n_rows;
      ```

    - 计算全表扫描**CPU代价**；
  - data_file_length： 聚簇索引总大小(非叶 + 叶)
  - index_file_length： 所有二级索引总大小
  - ...

- 收集策略
  - 表第一次open
  - analyze命令
  - 语句执行时

InnoDB层统计信息

- InnoDB层统计信息
  - 除了设置Server层统计信息外，还在本层维护了自身的统计信息
  - 根据此统计信息，计算全表扫描/索引扫描代价
- 主要统计信息
  - stat_n_rows
    - 表记录数量；I/U/D操作时，实时修改；
    - 用于设置MySQL Server层的records信息
  - stat_clustered_index_size
    - 聚簇索引页面总数量
    - 计算MySQL Server层，data_file_length信息
    - 计算全表扫描IO代价
  - stat_sum_of_other_index_size
  - stat_modified_counter
    - I/U/D，此值++
- 收集策略
  - 第一次open
  - stat_modified_counter取值：(> 2 000 000 000) or > (stat_n_rows/16)

InnoDB层统计信息

- 收集算法(老的)
  - 统计索引中叶页面数量
    - index->stat_n_leaf_pages
  - 随机定位索引中的8个叶页面
    - srv_stats_sample_pages = 8;
  - 统计页面中，前缀索引列组合的**Distinct数量**
    - 例如：Index idx (a, b, c)，包含3列
    - Distinct[a] = ? ; Distinct[a, b] = ? ; Distinct[a,b,c] = ?
  - 根据以上信息，计算
    - 表数据量
    - 每个索引前缀组合的**Distinct数量**
      - 用于计算MySQL Server层的**rec_per_key信息**
      - 是Join Optimizer最重要的统计信息

InnoDB层统计信息

- 统计信息持久化
  - –innodb_stats_persistent
    - **Introduced :** 5.6.6
    - **Default : ON**
  - –innodb_stats_persistent_sample_pages
    - Default: 20
  - – 信息存放的表
    - innodb_index_stats
    - innodb_table_stats
  - – 收集方式
    - innodb_stats_auto_recalc + records changed 10%
    - ANALYZE TABLE
    - update by manual if you are positive

Cost Based Oracle Fundamentals
Oracle 10053 trace

OPTIMIZER_TRACE的过程

Viewer　Text

```
JSON
  [] steps
    [0]
      join_preparation
        select# : 1
        [] steps
          [0]
            expanded_query : "/* select#1 */ select `c`.`customer_id` AS `customer_id`,`c`.`store_id` AS `store_id`,`c
    [1]
      join_optimization
        select# : 1
        [] steps
          [0]
            condition_processing
          [1]
            [] table_dependencies
          [2]
            [] ref_optimizer_key_uses
          [3]
            [] rows_estimation
          [4]
            [] considered_execution_plans
          [5]
            attaching_conditions_to_tables
          [6]
            [] refine_plan
    [2]
      join_execution
        select# : 1
        [] steps
```

join_preparation

- join_preparation
  - –初始化一些值并做权限检查
  - –expanded_query:把*扩展为表上的所有列
  - – 去除子查询中的冗余子查询
    - 去除IN/ALL/ANY/EXISTS子查询类型中子查询语句中的ORDER BY/DISTINCT/GROUP BY操作
  - – 预处理各种子查询
    - 转换子查询为半连接
    - 使用物化标识子查询
    - 执行IN向EXISTS转换
    - 执行 操作 ALL/ANY/SOME向MIN/MAX转换
    - 使用值代替标量子查询
  - –子查询优化

join_optimization

- join_optimization逻辑查询优化
  - 转换子查询为半连接
  - 消除外连接、消除嵌套循环
  - 条件表达式的优化
    - 等式合并
    - 常量求值(a=1+1等)
    - 条件去除(1=1等)
  - 全文检索优化
  - 优化带有聚集函数的子句

join_optimization

- 计算最优的查询计划
  - 初始化JOIN的数据结构，建立表之间的依赖
  - 基于连接信息更新依赖关系
  - 获取索引信息
  - 基于表的依赖关系选出半连接的表
  - 选出常量表，并获取真实数据
  - 为非常量表计算行数
  - 计算潜在的半连接物化操作的花费
  - 基于统计信息求解最好的连接顺序的花费

join_optimization

- 多表的连接顺序
  - N个表最多的连接顺序为N！,如有A B C三个表，可能的顺序有：
    - A → B → C
    - A → C → B
    - B → C → A
    - B → A → C
    - C → A → B
    - C → B → A
  - 连接算法
    - optimize_straight_join
    - find_best(5.6.X废弃)
    - greedy_search
  - 搜索深度
    - 连接的表数减去常量表数
    - 表太多得到的是目前最优的
    - 不推荐很多表(例如3表以上的关联)进行关联查询的原因

Cost估算实例

- mysql> desc customer;

Field	Type	Null	Key
customer_id	smallint(5) unsigned	NO	PRI
store_id	tinyint(3) unsigned	NO	MUL
first_name	varchar(45)	NO	
last_name	varchar(45)	NO	MUL
email	varchar(50)	YES	
address_id	smallint(5) unsigned	NO	MUL
active	tinyint(1)	NO	
create_date	datetime	NO	
last_update	timestamp	NO	

- mysql> show index from customer;

Table	Non_unique	Key_name	Column_name	Cardinality
customer	0	PRIMARY	customer_id	599
customer	1	idx_fk_store_id	store_id	4
customer	1	idx_fk_address_id	address_id	599
customer	1	idx_last_name	last_name	599

Cost估算实例

- select * from innodb_table_stats where table_name like 'customer%'\G;

```
            table_name: customer
           last_update: 2014-12-18 09:56:42
                n_rows: 599
    clustered_index_size: 5
sum_of_other_index_sizes: 3
```

- select * from tables where table_name='customer'\G

```
  TABLE_CATALOG: def
   TABLE_SCHEMA: sakila
     TABLE_NAME: customer
     TABLE_TYPE: BASE TABLE
         ENGINE: InnoDB
        VERSION: 10
     ROW_FORMAT: Compact
     TABLE_ROWS: 599
 AVG_ROW_LENGTH: 136
    DATA_LENGTH: 81920
MAX_DATA_LENGTH: 0
   INDEX_LENGTH: 49152
```

Cost估算实例

- mysql> select * from innodb_index_stats where table_name =' customer' ;

table_name	index_name	stat_name	stat_value
customer	PRIMARY	n_diff_pfx01	599
customer	PRIMARY	n_leaf_pages	4
customer	PRIMARY	size	5
customer	idx_fk_address_id	n_diff_pfx01	599
customer	idx_fk_address_id	n_diff_pfx02	599
customer	idx_fk_address_id	n_leaf_pages	1
customer	idx_fk_address_id	size	1
customer	idx_fk_store_id	n_diff_pfx01	2
customer	idx_fk_store_id	n_diff_pfx02	599
customer	idx_fk_store_id	n_leaf_pages	1
customer	idx_fk_store_id	size	1
customer	idx_last_name	n_diff_pfx01	599
customer	idx_last_name	n_diff_pfx02	599
customer	idx_last_name	n_leaf_pages	1
customer	idx_last_name	size	1

Cost估算实例

- 语句
  - SET OPTIMIZER_TRACE= “enabled=on” ,END_MARKERS_IN_JSON=on;
  - set optimizer_trace_max_mem_size=1024000;
  - select c.* from customer c where c.last_name=’JOHNSON ‘;
  - select * from information_schema.optimizer_trace \G

- 根据前面信息手工计算
  - clustered_index_size: 5
  - TABLE_ROWS: 599
  - Fetch data type: table_scan
  - cost= 5+599*0.2+1(微调)+ 1.1(微调)=126.9

- Trace里的信息

```
"range_analysis": {
  "table_scan": {
    "rows": 599,
    "cost": 126.9
  } /* table_scan */,
```

Cost估算实例

- 根据前面信息手工计算
  - idx_last_name_size: 1
  - n_diff_pfx01 : 599
  - clustered_index_size: 5
  - TABLE_ROWS: 599
  - Fetch data type: range_scan
  - cost= 1(回表1次)+1(索引)+1*0.2+?

- Trace里的信息

```
"ranges": [
  "JOHNSON <= last_name <= JOHNSON"
] /* ranges */,
"index_dives_for_eq_ranges": true,
"rowid_ordered": true,
"using_mrr": false,
"index_only": false,
"rows": 1,
"cost": 2.21,
"chosen": true
```

Cost估算实例

- 根据前面信息手工计算
  - idx_last_name_size: 1
  - n_diff_pfx01 : 599
  - clustered_index_size: 5
  - TABLE_ROWS: 599
  - Fetch data type: ref
  - cost= 1(回表1次)+1*0.2+?

- Trace里的信息

```
"best_access_path": {
"considered_access_paths": [
{
"access_type": "ref",
"index": "idx_last_name",
"rows": 1,
"cost": 1.2,
"chosen": true
},
```

Cost估算实例

- 最终的执行计划

```
"considered_execution_plans": [
{
"plan_prefix": [
] /* plan_prefix */,
"table": "`customer` `c`",
"best_access_path": {
  "considered_access_paths": [
    {
      "access_type": "ref",
      "index": "idx_last_name",
      "rows": 1,
      "cost": 1.2,
      "chosen": true
    },
    {
      "access_type": "range",
      "cause": "heuristic_index_cheaper",
      "chosen": false
    }
  ] /* considered_access_paths */
} /* best_access_path */,
"cost_for_plan": 1.2,
"rows_for_plan": 1,
"chosen": true
}
] /* considered_execution_plans */
```

- Trace里的信息

```
mysql> explain select c.* from  customer c where  c.last_name='JOHNSON';
+----+------+---------------+---------------+---------+-------+------+
| id | type | possible_keys | key           | key_len | ref   | rows |
+----+------+---------------+---------------+---------+-------+------+
|  1 | ref  | idx_last_name | idx_last_name | 137     | const |    1 |
+----+------+---------------+---------------+---------+-------+------+
```

总结

Trace信息是非常复杂的:

- 如果你读过源码，精通算法，比较容易读懂
- 很多地方有cost微调
- 不懂源码，可以先了解其思想，执行计划得出的大概过程。然后逐步看trace里的cost，反推计算方式，核实对应的统计信息是否正确

统计信息是非常重要的:

- 是CBO的基础
- 错误的统计信息会导致执行计划不准
- 执行计划错误的时候一般先检查统计信息

尽量写简单的sql:

- N个表最多的连接顺序为N!
- 关联表太多得到的执行计划只是部分最优的

Cost Based Oracle Fundamentals:

- Oracle 的CBO思想，对了解MySQL的CBO有很大的借鉴意义