An Overview of the MySQL 5.0 Query Optimizer

Timour Katchaounov, MySQL AB, developer

timour@mysql.com

Thanks to: Igor Babaev, Sergey Petrunia

MySQL Users Conference, 2005
April 18-21, 2005
Outline

• Background on query optimization
• Query execution plans in MySQL
• Cost-based optimization
• Controlling the optimizer
• New functionality and work in progress
What is this talk about ...

DBMS
- Automatic code generation
- Query execution

Some program

SQL

Level of abstraction

C/C++ program

DB Engine API

Storage manager

OS API

Data

C/C++ program

OS API
... the **MySQL optimizer** of course

![Diagram of DBMS components](image)

- **Query compiler**
  - **Parser** – analyze syntax
  - **Preprocessor** – semantic checking, name resolution
  - **Optimizer** – generate optimal Query Execution Plans
    - Query transformations
    - Search for optimal execution plan
    - Plan refinement

- **QEP**
- **Query execution engine**

---

- **DBMS**
- **SQL**
- **Parse tree**
- **Query Execution Plan**
Query execution plans – elements

Fixed set of operations:

- **Access methods** to retrieve data:
  - Table scan (**ALL**)
  - Index scan (**index**)
  - Index range scan (**range**)
  - Index by reference (**ref**)
  - Index by unique reference (**eq_ref**)
  - Constant (**const**)

- **Conditions** to filter data
  - <, >, =, etc.
  - AND, OR, (, )

- **Joins** to combine/relate data

- Other: union, group by, order by, etc.
Query execution plans – query execution plans as operator trees

A) Bushy plan

B) Left-deep (linear) plan
Example: World database

```sql
SELECT City.Name, Language
FROM Languages, Country, City
AND City.ID = Country.Capital
AND City.Population >= 5000000
AND Percentage > 4.0;
```
Query execution plans in MySQL—query plans as operator sequences

1) table = City
   condition = (Population >= 5000000)
   access = Index range scan
   index = Population, by
           (Population >= 5000000)

2) table = Country
   condition = (Country.Code = City.Country
              AND
              City.ID = Country.Capital)
   access = Index by unique reference
   index = Primary (Code), by
           (Country.Code = City.Country)

3) table = Languages
   condition = (Languages.Country =
              Country.Code
              AND Percentage > 4.0)
   access = Index by reference
   index = Primary (Country, Language), by
           (Languages.Country = Country.code)

Nested Loops Join

for each row_city =
    index_range(City, Population,
              "Population >= 5000000");

for each row_country =
    index_ref(Country, PRIMARY,
    if NOT (City.ID =
            Country.Capital)
        continue;

for each row_languages =
    index_ref(Languages,
              PRIMARY,
              "Languages.Country =
              Country.Code");
    if (Percentage > 4.0)
        send_result;

<row_city, row_country, row_languages>
Cost-based query optimization

• General idea:
  – assign **cost** to operations
  – assign **cost** to (partial) plans
  – search for plans with lowest cost

• Possible because:
  – query plans are simple
  – there is data statistics
  – there is meta-data

• Main characteristics of an optimizer:
  – search space
  – cost model
  – search procedure
Cost model

- Cost \sim disk accesses
- Cost unit = random read of a data page (4 Kb)
- Main cost factors
  - Data statistics:
    - number of pages per table (index) \sim P(R) (or P(I))
    - cardinality of tables/indexes \sim N(R)
    - length of rows and keys
    - key distribution
  - Schema:
    - uniqueness (PK), nullability
- Simplified cost model (table scan):
  - cost(access(R)) \sim P(R)
  - cost(R join S) \sim P(R) + N(R) \times P(S)
Optimizer search procedure (1)

• Goal – find optimal complete plan
• Complete plan = sequence of operators, for each operator fixed:
  – table / index(es)
  – access method
  – conditions
• Naïve search:
  – find the best access method for each table
  – generate/cost all operator permutations
  – select the best one
  – bad idea => $N!$ plans ($N$ – no. of tables)
Optimizer search procedure (2) – exhaustive search in MySQL

- Search all plans “bottom-up”:
  - begin with all 1-table plans
  - for each plan QEP = \(<T_1, ..., T_{k-1}>\)
  - expand QEP with remaining \(\{T_k, ..., T_n\}\) tables
    - if \(\text{cost}(<T_1, ..., T_k>) < \text{cost}(\text{best complete plan so far})\)
- Use the principle of optimality for pruning:
  “The sub-plans of an optimal plan are optimal for their size”
- The procedure “walks” over a search tree \(\Rightarrow\)
Optimizer search procedure (3) – depth-first search illustrated

SELECT City.Name, Language FROM Languages, Country, City
    AND City.ID = Country.Capital
    AND City.Population >= 5000000
    AND Percentage > 4.0;
remaining_tables = \{ t_1, \ldots, t_n \}; /* all tables in a query */

procedure find_best(
    pplan, /* in, partial plan of tables-joined-so-far \langle T_1, \ldots, T_k \rangle */
    pplan_cost, /* in, cost of pplan */
    remaining_tables, /* in, set of tables not referenced in pplan */
    best_plan_so_far, /* in/out, best plan found so far */
    best_plan_so_far_cost) /* in/out, cost of best_plan_so_far */
{
    for each table T from remaining_tables
    {

01) access_method(T) = best_access_method(T, pplan); /* Best access method from T to pplan. */
02) pplan_cost = cost(pplan, access_method(T)); /* Cost of the expanded plan. */
    if (pplan_cost \geq best_plan_so_far_cost)
03)        continue; /* Prune this branch of the search space. */
04) pplan = \langle pplan, access_method(T) \rangle /* Expand pplan by access_method(T). */
05) remaining_tables = remaining_tables - T;
06) if (remaining_tables is not an empty set)
    {
07)        find_best(pplan, pplan_cost, remaining_tables, best_plan_so_far, best_plan_so_far_cost);
    }
else /* Found a complete plan that is the best so far. */
    {
08)    best_plan_so_far_cost = pplan_cost;
09)    best_plan_so_far = pplan;
    }
}
The cost of optimization

• Exhaustive search with pruning still $O(N!)$
  – for $N > 13-14$ => several hours/days
  => exhaustive search infeasible for big queries

• “Greedy search” in MySQL 5.0
  – control how exhaustive – search depth
  – each search step:
    • estimate each potential extension up to search depth
    • pick the best extension => “greedy”
    • “forget” other extensions
    • continue with remaining tables
  – may miss the best plan => trade off
The “greedy” search procedure (1)

Current best plan:

1) < >

2) <T3>

3) <T3,T2>

4) <T3,T2,T4>

5) <T3,T2,T4,T1> - complete plan

Search tree:
The “greedy” search procedure (2)

remaining_tables = \{t_1, \ldots, t_n\}; /* all tables in a query */

procedure greedy_search (remaining_tables, search_depth) {
    pplan = < >;
    do {
        (t, a) = best_extension(pplan, remaining_tables, search_depth);
        pplan = concat(pplan, (t, a));
        remaining_tables = remaining_tables - t;
    } while (remaining_tables != {})
    return pplan;
}

best_extension ==> Modified find_best(..., search_depth);
Controlling the optimizer

- Control of access method selection
  - use index
    SELECT * FROM City USE INDEX (Country), ...
  - force index
    SELECT * FROM City FORCE INDEX (Country), ...
  - ignore index
    SELECT * FROM City IGNORE INDEX (Population), ...

- Control of join optimization
  - degree of exhaustiveness - “optimizer_search_depth”
    0 => automatic; 1 => minimal; 62 => maximal (default)
  - pruning - “optimizer_prune_level”
    0 => exhaustive; 1 => heuristic (default)
  - force join order - STRAIGHT_JOIN
    SELECT * FROM (CountryLanguage STRAIGHT_JOIN City) STRAIGHT_JOIN Country WHERE ...;
Other new optimizer/engine features in MySQL 5.0

• Nested outer joins
  – fully pipelined execution (no materialization)
  – by Igor Babaev

• Equality propagation
  – deduce new equi-join predicates
  – by Igor Babaev

• Index merge
  – “OR” of ref and/or range for different indexes
  – by Sergey Petrunia

• Optimization of MIN/MAX with GROUP BY
  – use index to find MIN/MAX values in groups
  – by Timour
Work in progress

- Hash-join
- Multi-range read handlers for MyISAM / InnoDB
- Full support for NATURAL/USING joins
- Batched key access
- Smart explain

- Many other improvements ...