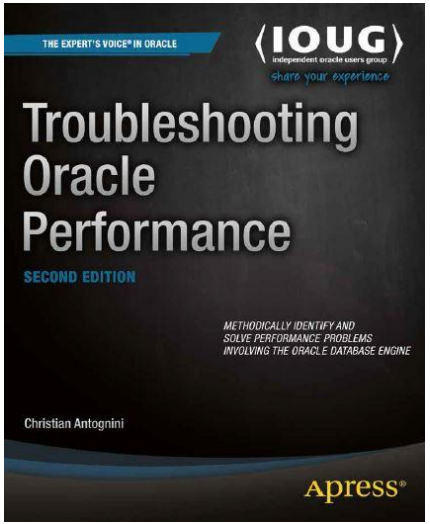
Query Optimizer – MySQL vs. PostgreSQL

Percona Live, Frankfurt (DE), 7 November 2018

Christian Antognini

@ChrisAntognini antognini.ch/blog

@ChrisAntognini



Senior principal consultant, trainer and partner at Trivadis

- christian.antognini@trivadis.com
- http://antognini.ch

Focus: get the most out of database engines

- Logical and physical database design
- Query optimizer
- Application performance management

Author of *Troubleshooting Oracle Performance* (Apress, 2008/14)

OakTable Network, Oracle ACE Director

trivadis
makes IT easier.

Agenda

1. Introduction
2. Controlling the Query Optimizer
3. Statistics about Data
4. Data Dictionary Metadata
5. Single-Table Access Paths
6. Joins and Sub-queries
7. Conclusion
8. References

Introduction

Compared Products

MySQL Community Server 8.0.13

Release date: 10 October 2018

Only the InnoDB engine is covered

PostgreSQL 11.0

Release date: 18 October 2018

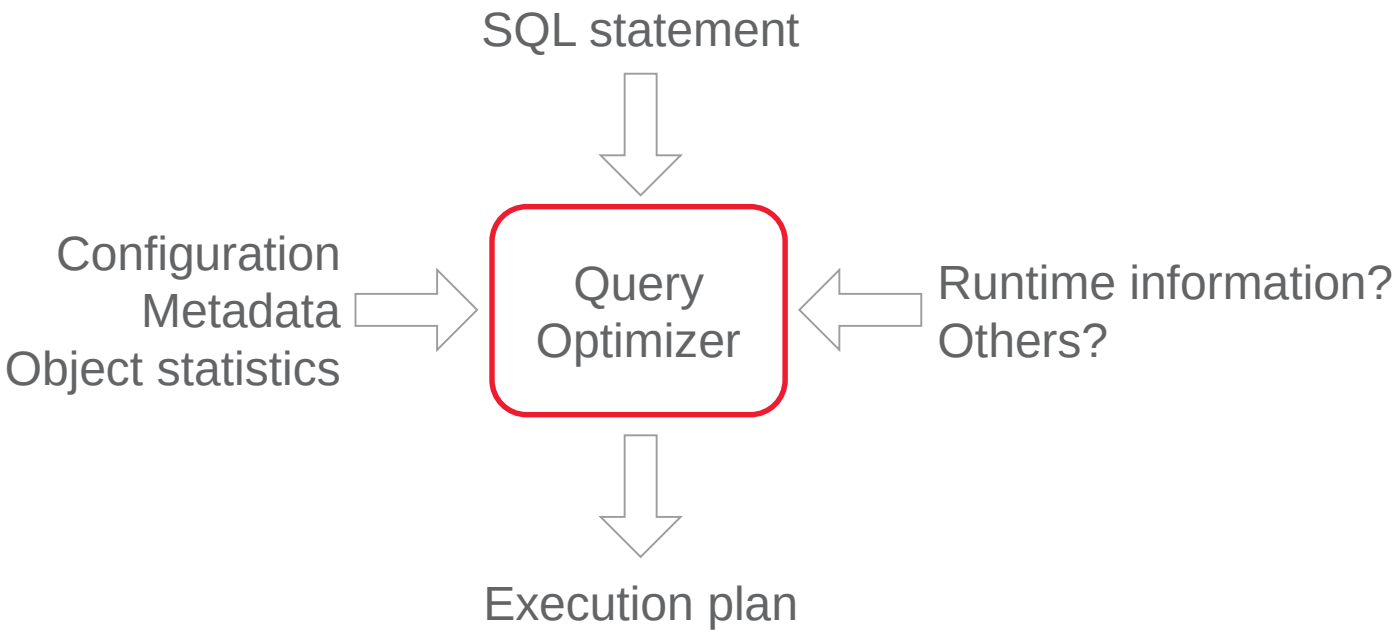
Disclaimer

No performance tests were performed

No comparison between the execution plans generated by the two query optimizers were performed

To compare and to evaluate the two query optimizers only the availability of key features and the ability of the query optimizer to correctly recognize common data patterns was considered

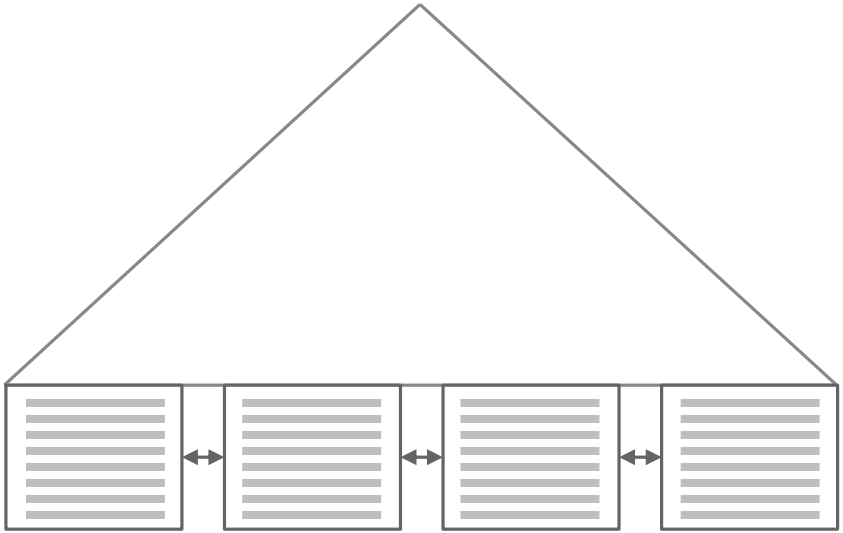
Inputs Considered to Produce an Execution Plan



How Is Data Stored?

MySQL

InnoDB uses a B-tree index



PostgreSQL

Heap table

Controlling the Query Optimizer

Configuration

MySQL

Three system variables control the behavior of the query optimizer

- Limit the number of evaluated plans (2)
- Control specific features (1 parameter for 21 features)

The default (system) values can be overwritten at session and statement level

PostgreSQL

45 parameters control the behavior of the query optimizer

- Limit the number of evaluated plans (2)
- Control specific features (25)
- Control the genetic optimizer (7)

The default (system) values can be overwritten at session level

Configuration – Cost Model

MySQL

A cost model database contains cost estimate information for a number of operations (8)

The default values can be changed at the system level only

PostgreSQL

A number of parameters provide cost estimate information for a number of operations (11)

The default (system) values can be overwritten at session level

Hints

hints_modifiers.sql
hints_index.sql
hints_optimizer.sql

MySQL

SELECT statement modifiers (4)

- Impact statement syntax

Index hints (3)

- Impact statement syntax
- Cause error when index missing

Optimizer hints (23)

- Similar to Oracle Database hints
- Global, query block and object-level
- Cause warning when syntax is wrong

PostgreSQL

(Available in *EDB Advanced Server*)

Statistics about Data

Gathering Statistics

MySQL

The ANALYZE statement gathers and, by default, stores statistics in the data dictionary

By default, an asynchronous automatic statistics recalculation takes place

Persistent (default) as well as non-persistent statistics exist

PostgreSQL

The ANALYZE statement gathers and stores statistics in the data dictionary

By default, the autovacuum daemon recalculate statistics of modified tables

Table Statistics

MySQL

Clustered index size (pages)

Number of rows

PostgreSQL

Table size (pages)

Number of rows

Number of pages marked all-visible

Column Statistics

MySQL

Data distribution (optional)

- Including fraction of entries that are null

PostgreSQL

Fraction of values that are null

Average column width (bytes)

Number of distinct values

Statistical correlation between physical and logical row ordering

Data distribution (optional)

Most common values and their frequency (optional)

Cross-Column Statistics

MySQL

PostgreSQL

Functional dependencies (optional)

Number of distinct values (optional)

Index Statistics

MySQL

Index size (pages)

Number of leaf pages

Number of distinct keys

- Several values are stored
- E.g. for index “a,b,c”
  → “a”, “a,b”, “a,b,c”, “a,b,c,PK”

PostgreSQL

Index size (pages)

Number of indexed rows

Data Dictionary Metadata

Constraints – Primary Key and Unique Key

MySQL

Because of the clustered index, PK has precedence over other indexes

Predicates based on UK take precedence over non-UK indexes

Equality predicates based on PK/UK are probed

PostgreSQL

No particular precedence is given to predicates based on PK/UK

Statistical correlation between physical and logical row ordering determines which index is used

Constraints – Foreign Key

MySQL

No usage of FK to avoid loss-less joins has been observed

PostgreSQL

No usage of FK to avoid loss-less joins has been observed

Constraints – NOT NULL

MySQL

NOT NULL constraints are used to verify the validity of predicates

PostgreSQL

By default the usage of NOT NULL constraints to verify the validity of predicates is enabled for specific cases only

- constraint_exclusion = partition
- Statistics are used instead

Constraints – CHECK

MySQL

No usage of CHECK constraints to verify the validity of predicates has been observed

- Statistics are used instead

PostgreSQL

By default the usage of CHECK constraints to verify the validity of predicates is enabled for specific cases only

- constraint_exclusion = partition
- Statistics are used instead

Single-Table Access Paths

Available Index Types

indexes_expression.sql
indexes_invisible.sql
indexes_non_key.sql
indexes_nulls.sql
indexes_partial.sql
indexes_prefix.sql

MySQL

Supported index types

- B-tree (default)
- R-tree (for spatial indexes)

Indexes can be created on expressions and, for string columns, on the leading part of column values

B-tree indexes store NULL values

Support for invisible indexes

PostgreSQL

Supported index types

- B-tree (default)
- Hash, GiST, SP-GiST, GIN, BRIN

Indexes can be created on expressions as well as on a subset of the rows

B-tree indexes store NULL values and support non-key columns

trivadis makes IT easier.

Optimization of ORDER BY, MIN and MAX

indexes_order_by.sql
indexes_min_max.sql

MySQL

B-tree indexes can be used to optimize ORDER BY, MIN and MAX

Index scans can be performed in both directions

Keys are stored according to the specified order

- NULLS FIRST/LAST not supported

PostgreSQL

B-tree indexes can be used to optimize ORDER BY, MIN and MAX

Index scans can be performed in both directions

Keys are stored according to the specified order

- NULLS FIRST/LAST supported

Merging Indexes

MySQL

Two or more B-tree indexes can be merged at runtime to evaluate multiple predicates combined with AND or OR

PostgreSQL

When appropriate, B-tree indexes are dynamically converted to bitmaps in memory

One utilization of this feature is to merge indexes to evaluate multiple predicates combined with AND or OR

(Declarative) Partitioning

partitioning_hash.sql
partitioning_list.sql
partitioning_range.sql
partitioning_sub.sql

MySQL

Available methods:

- Multi-column range and list
- Single-column hash
- Sub-partitioning by hash

Only local indexes (incl. PK/UK)

FK not supported

Partition pruning

PostgreSQL

Available methods:

- Multi-column range and hash
- Single-column list
- Sub-partitioning by range/hash/list

Only local indexes (incl. PK/UK)

FK cannot reference a partitioned table

Partition pruning

Joins and Sub-queries

Available Kind of Joins

joins_methods.sql
joins_order.sql
joins_syntax.sql
joins_bushy.sql

MySQL

Available join methods

- (Block) Nested loops join
- (Hash join available in MariaDB)

Bushy plans are considered only when no other possibility exists

Full outer joins are not supported

PostgreSQL

Available join methods

- Nested loops join
- Hash join
- Merge join

Bushy plans are considered

Full outer joins are supported and optimized with hash/merge joins

Sub-queries in WHERE Clause

MySQL

Simple sub-queries that are not correctly optimized were observed

- For optimal performance a rewrite might be necessary

Problematic cases observed

- Correlated NOT IN
- Correlated (NOT) EXISTS

PostgreSQL

Simple sub-queries that are not correctly optimized were observed

- For optimal performance a rewrite might be necessary

Problematic case observed

- Correlated (NOT) IN
- Uncorrelated NOT IN

Conclusion

Summary

MySQL	PostgreSQL
Good configuration capabilities	Good configuration capabilities
Hints available	Hints missing
Fairly good object statistics	Very good object statistics
Metadata only partially used	Metadata only partially used
Good indexing capabilities	Good indexing capabilities
Average partition capabilities	Average partition capabilities
Limited join capabilities	Good join capabilities

Core Messages

The query optimizer of PostgreSQL is more advanced than the one of MySQL

In general, the query optimizer of MySQL can only do a good job with transactional loads; the one of PostgreSQL is also suitable for analytical loads

References

References (1)

MySQL

[MySQL 8.0 Reference Manual]

[The Unofficial MySQL 8.0 Optimizer Guide]

[MySQL Internals]

[MySQL Server Blog]

PostgreSQL

[PostgreSQL 11 Documentation]

Planner source code “readme”

[PostgreSQL Wiki]

References (2)

The verification scripts I wrote are available on GitHub

How Well a Query Optimizer Handles Subqueries?

Q&A

trivadis
makes IT easier.